Action recognition using deep learning

Petar Palasek

Submitted in partial fulfillment of the requirements of the Degree of Doctor of Philosophy

Supervisor: Dr. Ioannis Patras

School of Electronic Engineering and Computer Science

Queen Mary University of London

United Kingdom

June 2017

Statement of originality

I, Petar Palasek, confirm that the research included within this thesis is my own work or that where it has been carried out in collaboration with, or supported by others, that this is duly acknowledged below and my contribution indicated. Previously published material is also acknowledged below.

I attest that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge break any UK law, infringe any third party's copyright or other Intellectual Property Right, or contain any confidential material.

I accept that the College has the right to use plagiarism detection software to check the electronic version of the thesis.

I confirm that this thesis has not been previously submitted for the award of a degree by this or any other university.

Signature: Petar Palasek

Date: 02/06/2017

Details of collaboration and publications:

- Palasek, Petar, and Ioannis Patras. "Discriminative convolutional Fisher vector network for action recognition." arXiv preprint arXiv:1707.06119 (2017).
- Marras, Ioannis, Petar Palasek, and Ioannis Patras. "Human Pose Estimation By Using Constrained Markov Random Fields As Convolutional Neural Networks." IEEE International Conference on Computer Vision (ICCV). IEEE, 2017.

- Marras, Ioannis, Petar Palasek, and Ioannis Patras. "Deep Refinement Convolutional Networks for Human Pose Estimation." Automatic Face and Gesture Recognition (FG), 12th IEEE International Conference on. IEEE, 2017.

- Tao, Ye, Petar Palasek, and Ioannis Patras. "Background Modelling Based on Generative Unet." 14th IEEE International Conference on Advanced Video and Signal based Surveillance (AVSS). IEEE, 2017.

- Palasek, Petar, and Ioannis Patras. "Action Recognition Using Convolutional Restricted Boltzmann Machines." Proceedings of the 1st International Workshop on Multimedia Analysis and Retrieval for Multimodal Interaction. ACM, 2016.

- Palasek, Petar, Heng Yang, Zongyi Xu, Navid Hajimirza, Ebroul Izquierdo, and Ioannis Patras. "A flexible calibration method of multiple Kinects for 3D human reconstruction." Multimedia & Expo Workshops (ICMEW), 2015 IEEE International Conference on. IEEE, 2015.

Abstract

In this thesis we study deep learning architectures for the problem of human action recognition in image sequences, i.e. the problem of automatically recognizing what people are doing in a given video. As unlabeled video data is easily accessible these days, we first explore models that can learn meaningful representations of sequences without actually having to know what is happening in the sequences at hand. More specifically, we first explore the convolutional restricted Boltzmann machine (RBM) and show how a stack of convolutional RBMs can be used to learn and extract features from sequences in an unsupervised way. Using the classical Fisher vector pipeline to encode the extracted features we apply them on the task of action classification. We move on to feature extraction using larger, deep convolutional neural networks and propose a novel architecture which expresses the processing steps of the classical Fisher vector pipeline as network layers. By contrast to other methods where these steps are performed consecutively and the corresponding parameters are learned in an unsupervised manner, defining them as a single neural network allows us to refine the whole model discriminatively in an end to end fashion. We show that our method achieves significant improvements in comparison to the classical Fisher vector extraction chain and results in a comparable performance to other convolutional networks, while largely reducing the number of required trainable parameters. Finally, we explore how the proposed architecture can be modified into a hybrid network that combines the benefits of both unsupervised and supervised training methods, resulting in a model that learns a semi-supervised Fisher vector descriptor of the input data. We evaluate the proposed model at image classification and action recognition problems and show how the model's classification performance improves as the amount of unlabeled data increases during training.

Acknowledgments

First of all, I would like to thank my supervisor Yiannis for all the help I received during the past four years of my PhD, for pushing me in the right directions, for helping me organize my thoughts and for understanding my concept of deadlines.

Next, I need to thank the people that I spent most of my time with in London, that are my flatmates Daria, who introduced me to London and showed me where Big Ben and Queen Mary are and how the Oyster card works, and Matija, my second flatmate who forced me to cook dinner multiple times a week, this way making me gain a skill that will help me survive wherever I end up next.

Then I want to thank all of my colleagues at QM, especially all of the members of the coffee gang (Yoshi, Sertan, Vangelis, Saverio, Stefano, Oya, Cesar, Aria, Juan, Mina, Young and the other Yiannis), the tea gang (Wenxuan and Ye), the members of the lunch group (Chris, Ivan, Fiona, Sally, Ellie, Andrej, Richard) and the rest of group (Faranak, Zongyi, Nur, Shenglan, Farzad, John, Krishna, Tomas, Vignes, Ning, Camilo) for making the day to day life at the lab interesting. I shouldn't forget to thank the people from other groups (Thomas and Dima, Léna, Maria, Katerina, Pablo and the rest of C4DM) for the moments outside of QMUL. Also, I would like to thank Ebroul for bringing us all in MMV together.

The next thanks goes to my family and friends back home who checked regularly how I was doing and when I was coming back again. Special thanks to those who also came here to visit me!

Of course, I also want to say thank you to you, for reading my thesis or at least the acknowledgments part of it, you're the best!

Contents

List of Abbreviations

BOW - bag of words

CNN - convolutional neural network

convRBM - convolutional RBM

CPU - central processing unit

DBN - deep belief network

EM - expectation-maximization

FC - fully connected

FPS - frames per second

FV - Fisher vector

GEM - generalized expectation-maximization

GMM - Gaussian mixture model

GPU - graphics processing unit

HOF - histogram of optical flow

HOG - histogram of oriented gradients

MBH - motion boundary histogram

OF - optical flow

PCA - principal component analysis

RBM - restricted Boltzmann machine

RGB - red green blue

SVM - support vector machine

Chapter 1

Introduction

Contents

1.1 Action recognition and deep learning

In 1966, Professor Marvin Minsky told one of his undergraduate students at MIT to connect a camera to a computer and make it describe what it sees. The fact that today, more than 50 years later, we still have thousands of scientists working on problems in the field that we now call computer vision, can definitely prove that the task assigned to the student as a summer project was not as easy as one might initially have thought. In fact, a lot of problems of the field have not been solved yet, including the problem of action recognition.

1.1.1 Action recognition

Before delving into action recognition as a problem in computer vision we first need to define what we mean when we talk about an *action*. There is a number of definitions

of an action offered in a recent survey [33], but in this thesis we will vaguely define it as the process of a person interacting with an object, one or more other people, or just performing body movements on their own. Some examples of actions could be a person running, walking, answering the phone, drinking from a cup, kissing another person, waving, riding a bicycle or reading a thesis. Action recognition is then defined as the problem of automatically recognizing the action happening in a given image or sequence of images.

The amount of video data accessible on the Internet is growing at extremely high rates; more than 400 hours of video data are being uploaded just to YouTube every single minute[1], as reported in 2015. Having this large amount of video data available online is a sufficient motivation for trying to develop methods that can learn meaningful representations from it and use them for the problem of action recognition. Being able to automatically recognize what the content of a given video, or, more narrowly recognize the actions that are depicted in it, is not only useful for organizing huge video datasets, but also something that could help improve systems for video surveillance, human-computer interaction systems and assistance systems.

Some examples of applications where action recognition can be used include: *content based video retrieval* (search the Internet for videos containing people juggling balls), *assistance systems for the elderly* (alert someone if a person falls at their home), *surveillance systems* (cut the power supply/stop the oncoming train if a person falls on the tracks) or *human-computer interaction* (pause the movie when the person watching it stands up and leaves their sofa). Some factors that make action recognition a challenging problem include handling occlusion, distinguishing between similar actions (e.g. jogging and running) or dealing with different camera motions and viewpoints. A similar problem to action recognition is action localization, where the task is to find the location at which the action is happening.

[1]Announced by YouTube CEO Susan Wojcicki on July 23, 2015 at VidCon conference.

1.1.2 Deep learning

Deep learning methods [10, 55] have been receiving a lot of attention in the recent few years because of the good performance they achieve in a variety of problems such as image classification, object detection, natural language processing, speech recognition and action recognition, just to name a few. A common property shared among deep learning architectures is that they are able to learn representations straight from the data which usually results in better performance than other methods that use handcrafted features.

Pinpointing the exact work that started the new raise in popularity of neural networks is difficult, but the work of [37] is definitely one of them. This work has helped solving a previously big problem of training larger neural networks by showing that a network with multiple hidden layers can be trained efficiently by building it in a layer-by-layer fashion [12]. Each layer is treated as a restricted Boltzmann machine [92, 28] trained using contrastive divergence [35, 36]. The networks built by stacking RBMs are called deep belief networks, but different kinds of deep learning architectures can be built by using different kinds of building blocks, such as autoencoders [12], denoising autoencoders [103], or convolutional RBMs [57]. The set of methods used for building these kind of architectures is referred to as deep learning [10, 4, 11]. The most important property of deep learning architectures is that they allow us to learn different levels of features [26], both in an unsupervised and a supervised way, starting from the simplest ones in the lowest levels and getting more and more complex as the number of layers increases. The same hierarchical, layer-wise architecture can be found in the visual cortex in the brains of mammals [42], which justifies deep learning being called a biologically inspired approach [34]. In fact, it has been shown that the lowest features learned using deep learning methods from sets of natural images correspond to neurons found in V1, the first stage of the mammal visual cortex [39], which can be seen as edge detectors.

The seminal work described in [50] has shown the power of models that can be trained end to end in a supervised way on large amounts of labeled data using back-propagation. This is one of the works that helped regain popularity of neural networks and start the deep learning revolution. Many works on deep networks have been published since then, not only for the task of image recognition [54, 18, 15, 90], but for the problem of action recognition [27, 45, 46, 89, 102] as well.

1.2 Problem definition

The main problem we wish to solve in this thesis is to recognize actions in sequences of images. To be more specific, we define the problem as follows. We are given an ordered sequence of images, that is a video clip of a few seconds of length, showing a person performing an action. The goal is to classify the action depicted in the previously unseen clip into one of the possible action categories, with examples of each category given in the training set.

1.2.1 Challenges and assumptions

The main challenges of the problem we are solving are the following. The videos that we want to classify are not of a fixed length, but their length can be in a range of only a few seconds up to a minute; usually the videos are in the order of tens of seconds long. Standard classifiers such as SVMs work with data of fixed length only, so our first goal is to extract a fixed length representation from videos of varying length in order to be able to classify them using standard classifiers. To this end we chose to use the Fisher vector encoding which we will discuss more in detail later.

Initial works on action recognition dealt only with videos taken in controlled environments, that is videos taken with no or a limited amount of camera motion, and usually with backgrounds that are simple or static. In this thesis we want to classify videos taken in unconstrained environments, meaning that the video we are classifying

may contain camera motion, different lightning conditions, occlusions, frames of low image quality, etc. Designing handcrafted features that would generalize well over all of the conditions mentioned above is difficult, so instead we explore methods in which the features used are learned from the data - using deep learning.

As mentioned at the beginning of the introduction chapter, there is a large amount of video data available on the Internet and it is growing rapidly every day. However, this data is not as tidy as the videos that are organized by the research community in the form of datasets made for evaluating different action recognition methods. The main difference is that most of the videos that can easily be found online are not labeled. Knowing that more data results in better performance when machine learning is involved, we want to find a way of leveraging this large amount of unlabeled video data. In order to try and learn meaningful representations from unlabeled videos that can later be used for the action recognition problem, we will try and design architectures that can be trained using unsupervised learning methods.

Another challenge when dealing with action recognition is that many videos depicting people performing an action from the same class can look very different when compared to each other. That is, there is a lot of intra-class variation present in the videos we deal with. The methods we use to learn representations from the videos should ignore the intra-class variations and focus on learning discriminative representations that make distinguishing between two different actions easier. We try to achieve this by proposing an architecture that can be trained in an end to end supervised way.

Similar to the problem of not having labeled video data which we mentioned above, but slightly different, is the problem when there is a fraction of the videos that are labeled and the rest is given without corresponding labels. We want to develop a technique that would allow us to make use of both labeled and unlabeled data, hoping that the large amount of unlabeled videos will increase the performance of our model. We try to solve this problem by proposing a semi-supervised method for training of

our architecture.

To summarize, we will deal with the action recognition problem using three different approaches, each of them based on the same idea; we aim to find a fixed length representation of the given sequence which we can then classify using a standard classifier such as an SVM. In the first approach we will try to extract the representation of the sequence in an unsupervised way, without making use of the action labels assigned to each of the sequences in the training set. As the second approach we will introduce a method that allows us to extract the representations in a supervised way, using the labels to help make the representation of the sequence become more discriminative and more suitable for the classifier which is trained at the same time, using the same learning objective. In the third approach we will deal with the problem of not having a training set with sequences that have all been labeled, but instead having only a limited set of labeled training data and a second set with unlabeled data. For this approach we introduce a semi-supervised training method.

The assumptions about the videos we work with in this thesis are the following. We assume that each video is depicting a person performing an action which can involve interacting with an object or objects, interacting with one or more people, or just performing body movements. We assume that the frame rate of the video is fixed and that only one action is shown in each given clip. As mentioned before, the clips do not have to be of the same duration.

Throughout the thesis we will distinguish between two different types of features; *static* features and *dynamic* features. What we consider static features are features that are extracted from a single or from multiple grayscale or RGB images. We may also mention features extracted from *static frames* in the thesis; this also refers to static features, we only wish to highlight that there is no motion information explicitly used when the features were extracted. On the other hand, we say that dynamic features are the features that are extracted in a way that explicitly makes use of precomputed

optical flow between frames. Both static and dynamic features can be *handcrafted* or *learned* straight from the input data.

1.3 Contributions

In this section we list the main contributions of the thesis. In the first main chapter we apply convolutional restricted Boltzmann machines in order to extract representations of videos that can be learned in an unsupervised way. The main contributions of that chapter can be listed as:

- We modify the energy function of the convolutional RBM model so that we can simply differentiate the objective function to arrive to the same learning updates as reported in [58]. The updates in that work do not follow directly from the energy function as it was defined, but are additionally normalized at the end. We include the needed normalizations explicitly in the energy function. We also show how to redefine the convolutional RBM in terms of the free energy function by marginalizing out the hidden units in the equation describing the probability of a sample. Having an objective function defined in such a way allows us to use libraries that support automatic differentiation such as Theano [100] when implementing the model, also making it possible to run experiments on a GPU.

- We show that the representations that are derived from unsupervised training of the convolutional RBMs have a better descriptive power than handcrafted image descriptors and give competitive performance in the problem of action recognition. We show this by running experiments on the standard UCF-101 action recognition dataset [93].

In the following chapter we reformulate the classic Fisher vector pipeline in terms of network layers, allowing us to form a convolutional Fisher vector network which

can be trained in an end to end supervised way. We summarize this chapter's main contributions as:

- We describe a novel neural network architecture for action recognition which includes two new types of layers; the Gaussian mixture model layer and the Fisher vector descriptor layer. Combining these layers with other standard ones into a single deep neural network gives us a way of jointly finetuning the parameters of the whole architecture with respect to a chosen discriminative cost, using the standard backpropagation algorithm. We show that adding supervision at every stage of the network improves the discriminative power of the extracted Fisher vector descriptor compared to the standard version of the descriptor which is extracted in an unsupervised manner.

- Analogous to convolutional neural networks, where the same operation is applied at different locations of the input tensor, our network offers a natural way of extracting the Fisher vector descriptors densely from a given input video, both in space and in time. This also allows us to easily extract the descriptor only from selected parts of the video, providing a straightforward way of implementing other architectures, such as spatial pyramids.

- We show that the proposed architecture can be used as a replacement for the fully connected layers in popular convolutional networks such as the VGG-16 network, achieving a comparable classification performance while reducing the total number of trainable parameters by a factor of 5.

In the third main chapter we introduce the semi-supervised Fisher vector network which can The chapter's main contributions can be summarized as:

- We describe a method for fitting Gaussian mixture models that can be used when the training data is not available all at once, but it is arriving in mini-batches,

i.e. the data is arriving in small subsets of the training set. The same method can be used for training of the GMM defined as a network layer, described in the previous chapter. Also, the method can be run on a GPU, leading to smaller training times compared to methods that can only run on CPUs.

- We define a method that leverages the availability of unlabeled data for training of an improved, semi-supervised version of the Fisher vector encoding. Similarly as in the previous chapter, we define a network that extracts the Fisher vector descriptor of the input data, which we name the semi-supervised Fisher vector network. We perform a number of experiments which show how increasing the amount of unlabeled data helps improve the classification performance of our model at the problem of image classification (on CIFAR-10) and action recognition (on UCF-101).

1.4 Outline of the thesis

The rest of the thesis is structured as follows. We start by briefly describing the related works in Chapter 2. We follow by describing the unsupervised approach of extracting video representations using convolutional restricted Boltzmann machines in the classic Fisher vector pipeline for action recognition in Chapter 3. Chapter 4 deals with reformulating the classic Fisher vector pipeline in terms of network layers, allowing us to build a convolutional Fisher vector network, trainable in a supervised, end to end way, which we also evaluate on the problem of action recognition. Chapter 5 introduces the semi-supervised Fisher vector network that leverages unlabeled data in order to improve the classification performance of the Fisher vector network in cases where the amount of labeled data is limited. We evaluate its performance on a image classification problem and an action recognition problem. Finally, we conclude with Chapter 6.

Chapter 2

Related work

Contents

Action recognition is an extensively studied research area of computer vision with a large and rapidly growing amount of works on the topic available in the literature. In this chapter we will briefly mention and describe the works that are closely related to what we do and refer the reader looking for a more detailed summary to recent survey papers such as [17], [118], [69] and [33]. The reasons for the popularity of action recognition as a research topic lies in the broad range of potential applications, some of which we have listed in the previous chapter; content based video retrieval, human-computer interaction, surveillance systems, assistance systems and others.

Many works on action recognition build upon works that have previously shown good performances in image recognition problems, by extending what worked in two dimensions to 3D, to also handle the time dimension. We will start by explaining some of the classical action recognition approaches that make use of handcrafted features and later move on to more recent approaches that employ deep learning methods.

2.1 Classical approaches

The classical image classification pipeline usually consists of the following steps; interest point detection, local descriptor extraction, local descriptor aggregation and classification. The same pipeline has been applied for the problem of action recognition following the seminal work of [51] on Space-Time Interest Points (STIPs). We continue by describing some approaches for each of the pipeline steps mentioned above.

2.1.1 Interest point detection

In images, the points that show a significant local variation of image intensities are referred to as interest points. The work of [51] introduces a 3D extension of the existing Harris corner detector [32], the Harris3D detector, and shows that it can be used to detect interesting events in videos. Arguing that the interest points detected by the Harris3D detector are too rare, the Gabor detector has been introduced in [24]. The Hessian detector has also been extended for detecting interest points in videos in [114]. An evaluation of different interest point detectors has been done in [109] on three standard action recognition datasets: KTH [86], UCF Sports [82] and Hollywood2 [63]. The conclusion of the evaluation was that often a simple dense sampling method, where interest points are sampled at equal distances in space and time, will results in the best performance. This, however, depended on the dataset used; in some cases the best performance was achieved using the Harris3D detector [69].

Trajectories

Closely related to detecting interest points in videos is the idea of tracking them through a number of consecutive frames in time, forming trajectories. This was explored in two works [66, 64] where the popular Kanade-Lucas-Tomasi (KLT) tracker [61] was applied in order to track the detected points. A different method was described in [95] where SIFT descriptors [60] were matched in consecutive frames to form trajectories. Knowing that extracting interest points by dense sampling often

leads to better performance in action recognition than when other interest point detectors are used [109], the same idea was interesting to explore when trajectories are used. However, densely matching SIFT descriptors is computationally expensive, so a different method, where densely sampled points are tracked using optical flow fields is introduced in [105]. A thorough evaluation of the different methods for extracting trajectories mentioned above was done in [106]. The dense trajectories outperformed all other trajectories on nine action recognition datasets. In [107], camera motion is estimated by calculating the homography between two consecutive frames by matching SURF descriptors [9] and using the RANSAC [29] algorithm. Removing trajectories consistent with camera motion led to improved dense trajectories, the most popular handcrafted representation used for action recognition [118]. Other works that use trajectories for the problem of action recognition include [43, 115, 6].

2.1.2 Local descriptor extraction

The next step following interest point detection is the step of extracting local descriptors around the detected points. The early works such as [51] and [24] extract the descriptors from cuboids, i.e. from cubes formed around the detected point in space-time, but the same can be applied to trajectories. Some of the most popular local descriptors that can be extracted both from cuboids or trajectories are described below.

A 3D version of the Histogram of Oriented Gradients (HOG) [20] descriptor is introduced as HOG3D in [47]. A spatio-temporal grid is formed around an interest point and for each of the cells a histogram descriptor is calculated. The histograms are concatenated into the final descriptor after being normalized [69]. The popular SIFT descriptor [60] was also extended for the purpose of being applied to video data in [87]. The work of [52] makes use of optical flow calculated between consecutive frames in the video and proposes the Histogram of Optical Flow (HOF) descriptor in order to characterize local motion of the space-time neighborhood around detected interest points. A more robust descriptor extracted from optical flow is the Motion

Boundary Histogram (MBH) introduced in [21]. Similar to the HOG descriptor, the MBH descriptor bins the orientation information of spatial derivatives calculated for the horizontal and vertical components of the optical flow into histograms. The MBH descriptor is considered

Trajectories can be described using a list of displacement vectors showing how each of the tracked points moves through time, followed by a normalization by the sum of displacement vector magnitudes, as it was done in [105]. All of the local descriptors mentioned above can also be extracted along trajectories. In [106], a space-time volume aligned with each trajectory is formed, divided into cells each of which is described by a combination of HOG, HOF and MBH descriptors extracted from it.

2.1.3 Local descriptor aggregation

The next step of the classical pipeline is to form the representation of a whole video, by aggregating the extracted local descriptors using one of the local descriptor aggregation methods. In this subsection we will mention some of the popular aggregation methods that have been used for the problem of action recognition, starting with the bag of words model, introduced in [84] for the problem of document retrieval. One of the problems that we overcome by aggregating local descriptors is that we end up with a fixed length representation of an image or a video, as the number of features extracted in the previous step can vary.

The bag of words model is often applied in computer vision related tasks, where it is referred to as bag of visual words [19] model. The idea of the bag of words model is to represents an image (or a sequence of images) as a normalized histogram of so called code-words. The code-words are formed at learning time by clustering similar image-patches described by a chosen feature descriptor. This way similar images will result in having a similar histogram, which can then be used as the input of a classifier such as an SVM. Some of the first BOW-based approached for action recognition include the

works of [24] and [73]. One drawback of BOW is that it does not model the space-time configuration of the detected code-words, thus ignoring a lot of information about the observed image sequence. One solution for this problem is proposed in [52], where the idea of spatio-temporal grids is introduced. Using BOW models with trajectories is one of more popular approaches for action recognition.

A different aggregation method is based on sparse coding [74]. The main idea of sparse coding is to form a set of overcomplete basis (called a dictionary) and use it to represent the essential information of a signal using a very small number of non-zero elements. In [119] local 3D spatial-temporal gradient features are encoded by transforming each local spatial-temporal feature to a linear combination of atoms from a trained dictionary. After that, maximum pooling is done over the whole set of sparse codes from the video to obtain the final representation of the input sequence. The sparsity of the obtained representation strengthens the discriminative power and helps improve the recognition accuracy. The work of [31] also explores the effectiveness of sparse representations obtained by learning a set of overcomplete basis for the problem of action recognition. Each descriptor is represented by some linear combination of a small number of learned dictionary elements.

Another popular feature aggregation technique is the Fisher vector descriptor, based on the principle of Fisher Kernels [40]. The Fisher vector descriptor is a global descriptor used to represent data by describing how the parameters of a generative model fitted on a distribution of local features should change in order to better model the local features extracted from the given data sample. More specific, the Fisher vector is the gradient of a given sample's likelihood with respect to the parameters of the distribution estimated on the training set, scaled by the inverse square root of the Fisher information matrix.

Fisher vector descriptors were firstly introduced for solving the problem of image classification in [79]. An introduction to the theory and practice of using Fisher vectors

for the task of image classification is described in [85]. The main difference between the Bag of Words model and the Fisher vector encoding is that the BOW uses hard-assignment when aggregating descriptors, while soft assignment is used in Fisher vectors. A detailed evaluation on Fisher vector based techniques for action recognition can be found in [75].

A simplified version of the Fisher vector encoding is VLAD, Vector of Locally Aggregated Descriptors [44, 3]. First a codebook of k visual words is learned using k-means. Local descriptors are then associated to their nearest visual words and the VLAD descriptor accumulates the differences of the descriptors and their corresponding nearest visual words. The VLAD descriptor does not store store second-order information in contrast to the Fisher vector descriptor which does. When a Gaussian mixture model is used to build the codebook, the assumption that all Gaussians have the same variance in VLAD descriptors has been shown to be detrimental to performance in a number of computer vision problems where Fisher vectors perform in general better.

A more detailed overview of different feature encoding methods can be found in for example [14] and [112]. After the local descriptor aggregation step the resulting representation can be used for classification, usually using a classifier such as a support vector machine.

2.2 Deep learning approaches

In this section we will describe some of the important deep learning models that were applied for either image classification or action recognition problems, focusing on the ones that are more relevant to our work described in this thesis.

The building block that we use in our architecture described in the first main chapter is based on the restricted Boltzmann machine (RBM [36], also known under different

names as the Harmonium [91] or the Combination model [30]) which is explained briefly here. An RBM is a generative graphical model that consists of two layers of stochastic units; visible and hidden units, as illustrated in Figure 2.1. The visible units represent the inputs to the model and the hidden units are used to model the dependencies between the visible units [28]. In contrast to the standard Boltzmann machine [2], the RBM does not allow any connections between units in the same layer, hence the attribute restricted. Each unit in the model is associated with a bias value which we denote with b for the hidden units and c for the visible units. There are

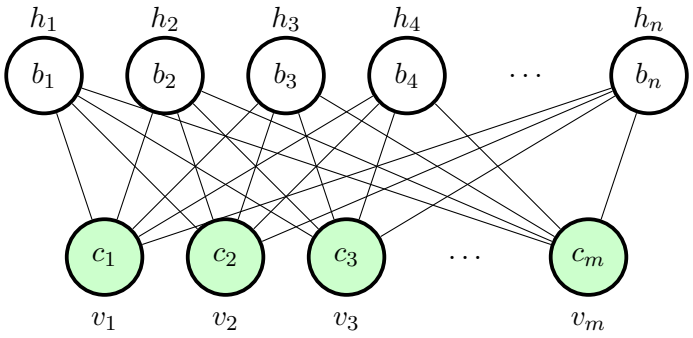


Figure 2.1: An illustration of an RBM with m visible and n hidden units.

different types of units that can be used in the visible or the hidden layer, but in the basic formulation they are both binary (Bernoulli variables). The RBM, or the joint configuration of the visible and hidden units $(\mathbf{v}, \mathbf{h})$, is described by an energy function $E_\theta(\mathbf{v}, \mathbf{h})$. The energy function changes depending on the types of visible and hidden units used; some examples can be found in [72, 36, 28, 113]. A useful property of RBMs is that they can be greedily trained and stacked on top of each other to form deep belief networks (DBNs) [37, 12] allowing us to learn representations of gradually increasing complexity. Training of these models using maximum likelihood learning is intractable, but they can be trained using contrastive divergence [35].

There is a problem applying RBMs or DBNs to image data - these models were not designed taking into account the fact that objects can appear at different locations

in images. By using them with image data the weights for detecting specific features would need to be learned for each location in the image separately, making the scaling of these models to bigger images difficult. To overcome this issue, a convolutional version of the RBM and the DBN is introduced in [57, 58], where the weights are shared over all the locations in the image. The idea of convolutional neural networks is not new, it goes back to the work of [56] where the famous LeNet-5 architecture for handwritten digit recognition was introduced. Convolutional RBMs can also be stacked into convolutional deep belief networks, resulting in learned filters of higher and higher complexity. Examples of the filters learned by stacking convolutional restricted Boltzmann machines are shown in Figure 2.2.

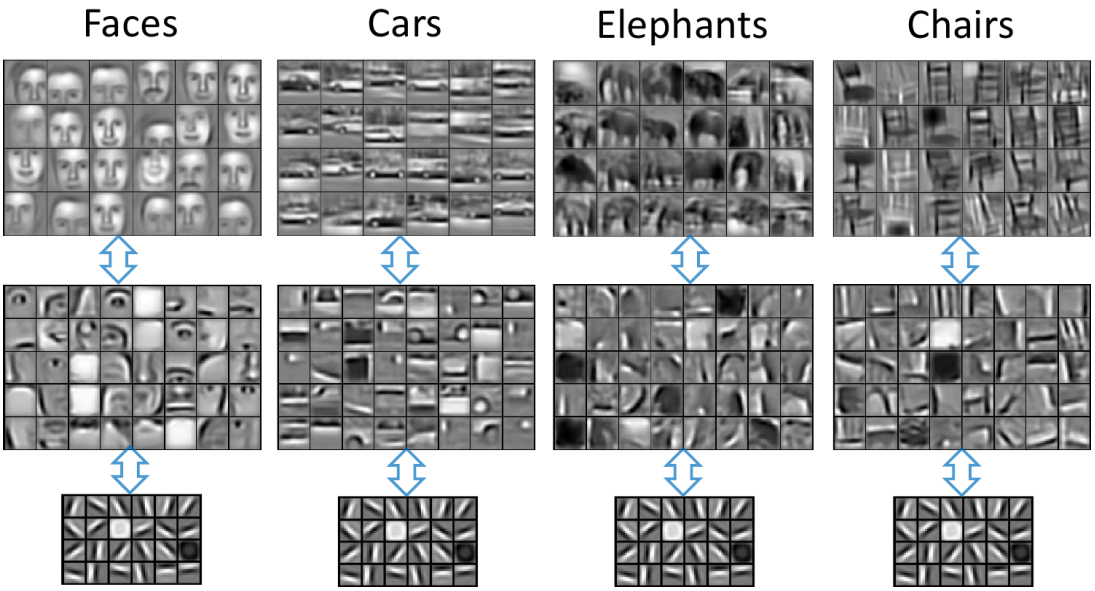


Figure 2.2: An illustration of different filters learned by stacking convolutional restricted Boltzmann machines borrowed from [57]. Starting from the simplest edge detectors in the first layer, the filters become more and more complex as more layers are added.

A number of extensions of deep learning architecture building blocks based on RBMs have been introduced in order to allow their usage on sequential data. For example, the temporal RBM introduced in [96] augments the standard RBM by adding connections from previous states of the visible and hidden units. As the RBM is a generative model, the temporal RBMs can be used to generate image sequences. In [65] the gated

RBM was introduced, which allows image patch transformations to be represented in the model's hidden units by forming three way connections between the input, hidden and output units. This was later extended in order to scale to realistic image sizes by introducing the convolutional GRBM [99], also following the idea of convolutional neural networks from [56]. One more model incorporating convolutions is the Space-time deep belief network described in [16]. It is a model that repeatedly performs space, followed by time pooling, based on the convolutional RBMs. The models based on RBMs are particularly interesting to us as they can be trained in an unsupervised manner, which is something we explore in the first main contribution chapter of the thesis.

The classical pipeline for image classification which we mentioned at the beginning of the chapter, consisting of interest point detection, local descriptor extraction, local descriptor aggregation and classification steps were proven to be replaceable by convolutional neural networks after the seminal work of [50]. The work built upon the idea of [56] and won the ImageNet challenge [83] by a large margin over the other approaches in the competition, showing that deep convolutional networks can be train using standard backpropagation given enough training data, suitable hardware (GPU) and proper regularization techniques, namely dropout. Dropout [38] is a simple method for regularizing neural networks, in which a percentage of neurons is randomly omitted during training, in this way preventing complex co-adaptations of feature detectors, helping prevent the overfitting problem.

The standard convolutional neural networks usually consists of a number of convolutional layers that contain a set of trainable filters that are convolved with the input to produce an output, optional normalization layers, followed by pooling layers that reduce the dimensionality of the preceding layer's output and a number of fully connected layers at the end of the network. The advances of neural networks are not just caused by the availability of larger datasets and faster hardware, but are the con-

sequence of designing new types or architectures and new algorithms [98]. The most straightforward way that helped improve the performance of a CNN was by increasing its size in both terms of depth and width. A thorough evaluation of the effects of increasing the networks size was performed in [90] and [15], where the CNN-M-2048 and VGG-16 networks were described which we will also use in our work. The details of these networks are shown in detail in Appendix C.

Following the growing popularity of deep neural networks, several works were published on using neural network based approaches for the problem of action recognition. In [45] a 3D extension of the standard 2D convolutional neural networks (CNNs) was introduced, where information from both the space and the time dimension are included by performing 3D convolution. A 3D convolution is also used in [7] results of which are then fed into a recurrent neural network for classification. A more recent work described in [102] also applied 3D CNNs, but with a much deeper network architecture. The work of [46] examines different kinds of extending CNNs into the time domain by fusing features extracted from stacks of frames in order to include motion information. Transfer learning is also applied in order to prevent overfitting to small video datasets. Motion information was included in the work of [89] in an explicit way by providing dense optical flow at the input of the network. More specifically, two streams of a network are employed; one performing classification based on static video frames and the other based on the optical flow. Different ways of fusing the spatial and the temporal streams of such networks are studied in [27]. The work of [116] considers different ways of aggregating strong CNN image features over long periods of time, including feature pooling and using recurrent neural networks.

2.3 Fisher vector based approaches

In this section we will mention more works that make use of the Fisher vector descriptor, in addition to the ones already mentioned in 2.1.3. The first work that

applied Fisher vector descriptors for the problem of action recognition in videos used HOG, HOF and MBH features [109] extracted along dense trajectories as local features [105]. The trajectories are extracted by defining a dense grid of points which are then tracked using optical flow that was estimated offline, this way including motion information in the pipeline. By encoding the extracted trajectory features with the Fisher vector descriptor, this approach and the improved version of [107] achieved state of the art results for the action recognition problem. Another work that uses features extracted from two-streams of deep CNNs along trajectories is [117].

Recently several works that try to combine the power of Fisher vector representations and neural network approaches have been published. This includes deep Fisher networks from [88] used for large-scale image classification, stacked Fisher vectors [78] used for action recognition, deep Fisher kernels [97] and the hybrid classification architecture [80] also applied on image classification problems. In [76] CNN features are extracted from random subvolumes of a video and encoded using the Fisher vector descriptor in order to arrive at a representation suitable for video classification.

The work of [88] combines ideas from the area of neural networks with the Fisher vector descriptor by forming a deep Fisher network in which two Fisher vector layers are stacked. The network is discriminatively trained for the problem of image classification, however the features at the input layer are fixed, manually-designed features. Stacked Fisher vectors are also applied for action recognition in [78], where the first layer encodes the improved dense trajectories from [107]. After discriminative dimensionality reduction a second Fisher vector encoding is done. The combination of the FV and the stacked FV showed to be beneficial. The work in [80] treats the Fisher vector descriptor as an unsupervised layer followed by a number of fully connected layers. that can be trained with backpropagation. End to end training of a Fisher kernel SVM viewed as a deep network is done in [97] for the problem of image classification. This method uses manually-designed features at the input layer and requires

retraining of the SVM on the whole training set at each step of the training.

2.4 Conclusion

In this chapter we have mentioned and briefly described some of the approaches used for the problem of action recognition. A summary of the state of the art results evaluated on the UCF101 action recognition dataset is available in Table B.1 in Appendix B.

In the first section we mentioned a number of classical methods that follow the pipeline borrowed from works on image classification consisting of steps for detecting (and tracking) interest points, extracting local descriptors, aggregating them and finally classifying the resulting representations. These works generally worked using handcrafted features; designing these is a difficult problem and requires a lot of expert knowledge and they often do not generalize well for different tasks.

We followed by describing some of the popular deep learning architectures that have the ability to learn representations from the data itself, without the need for a human expert. Many works show that methods that use learned representations perform better than the ones that are handcrafted, so we plan to follow this approach in our work. Having identified the problem of having large amounts of unlabeled video data available we want to try developing architectures that can learn meaningful representations of videos it in an unsupervised way. This, and the related problem of using both unlabeled and labeled data in the problem of action recognition will be two of our main points that we will focus on in this thesis.

We recognized the power of the Fisher vector encoding used by many works on action recognition. Even though some of the recent works explore adding supervision at different steps of the standard Fisher vector descriptor pipeline, none of them have tried to refine the local feature extraction, Fisher vector encoding and the classification steps jointly. We will also explore this idea in the thesis.

Chapter 3

Action recognition using convolutional restricted Boltzmann machines

Contents

In this chapter we will try to extract a meaningful representation of an image sequence in a unsupervised way and then use it for the problem of action recognition. To this end, we will focus on using a generative neural network, namely the convolutional extension of a restricted Boltzmann machine (RBM). We first use a stack of convolutional restricted Boltzmann machines to learn and extract features from sequences of images in an unsupervised way, aggregate them into a Fisher vector representation which we then classify into action classes using an SVM. We modify the energy function of the convolutional RBM in such a way that the training updates reported in the literature follow directly from the differentiation of the objective function, which we define in terms of the free energy function. This is in contrast to other works on con-

volutional RBMs in the literature whose update equations do not directly follow from a well defined energy function or optimization framework without any ad hoc normalizations. Having an objective function defined in such a way will allow us to easily use it in libraries that provide automatic differentiation and support for running code on GPUs. We show that the representations that are derived from unsupervised training of the RBMs have very similar or better descriptive power than handcrafted image descriptors and give competitive performance in the problem of action recognition.

The main contributions of this chapter can be summarized as follows:

- We modify the energy function of the convolutional RBM model so that we can simply differentiate the objective function to arrive to the same learning updates as reported in [58]. The updates in that work do not follow directly from the energy function as it was defined, but are additionally normalized at the end. We include the needed normalizations explicitly in the energy function. We also show how to redefine the convolutional RBM in terms of the free energy function by marginalizing out the hidden units in the equation describing the probability of a sample. Having an objective function defined in such a way allows us to use libraries that support automatic differentiation such as Theano [100] when implementing the model, also making it possible to run experiments on a GPU.

- We show that the representations that are derived from unsupervised training of the convolutional RBMs have a better descriptive power than handcrafted image descriptors and give competitive performance in the problem of action recognition. We show this by running experiments on the standard UCF-101 action recognition dataset [93].

3.1 Convolutional restricted Boltzmann machine

In this section we will describe and modify the convolutional extension of the RBMs in such a way that the training updates reported in the literature follow directly from the differentiation of the objective function, which we define in terms of the free energy function. This is in contrast to other work on convolutional RBMs (convRBMs) where the updates do not directly follow from their energy function.

The drawback of RBMs and DBNs is that they do not address the fact that objects can appear at different locations in images. This means that the weights for detecting specific features need to be learned for each location in the image separately, making the scaling of these models to bigger images a difficult problem. To overcome this issue, a convolutional version of the RBM and the DBN is introduced in [57, 58], where the weights are shared over all the locations in the image. In this chapter we will use the convolutional RBM model defined by [57], but with a modified energy function which we describe in the following section. An illustration of the convRBM is shown in Figure 3.1.

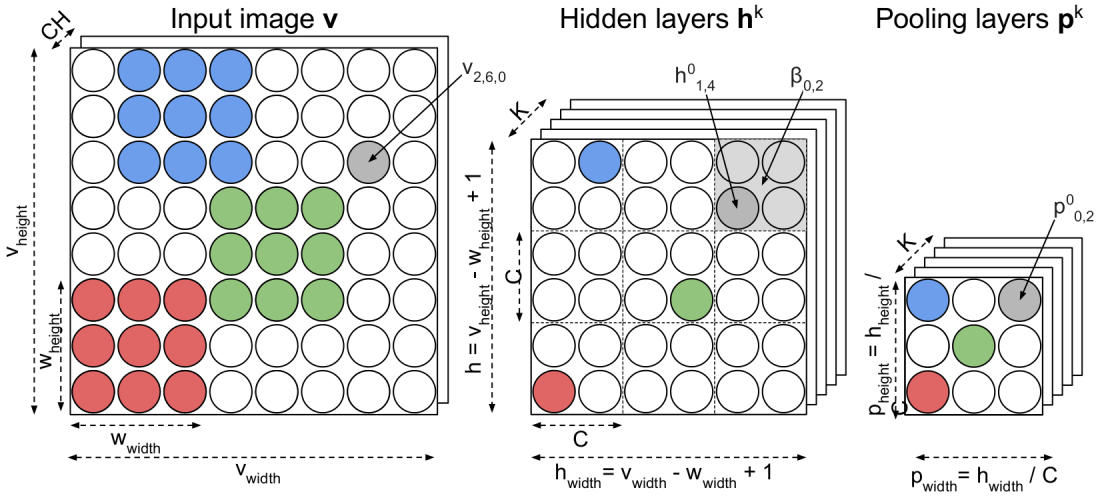


Figure 3.1: Illustration of the convolutional RBM with probabilistic max-pooling.

3.1.1 Modified energy function

Originally, the energy function of a convRBM with binary visible units is defined as [57, 58]:

$$E_\theta(\mathbf{v},\mathbf{h}) = -\sum_k \sum_{i,j} \left(h^k_{i,j} \left(\tilde{\mathbf{W}}^k *_v \mathbf{v} \right)_{i,j} + b_k h^k_{i,j} \right) - c \sum_{i,j} v_{i,j}, \tag{3.1}$$

subject to $\sum_{(i,j)\in B_\alpha} h^k_{i,j} \leq 1, \forall k, \alpha$,where $\mathbf{v}$ is the input, $h^k_{i,j}$ is a unit in the hidden layer, $\mathbf{W}^k$ is a filter, b_k and c are the hidden and visible biases and B_α is a group of hidden units. In our work we want to be able to model real-valued data so we use the Gaussian-Bernoulli version of the convRBM. Changing the visible units from binary to Gaussian units is done by adding a quadratic term to the energy function, as in e.g. [58, 59] and by scaling the visible unit terms by the variance σ^2. Similar modifications of the RBM energy in order to deal with real-valued data are done in [36]. We further modify the energy function by normalizing the sums over the visible units and by adding a root term that will end up as a normalization factor once we define the model in terms of the free energy function. Including those in the energy function is what makes the updates reported in [58] follow from the objective function which we define later. Additionally, we split the term summing over the hidden units into two sums, as it made the derivation of the learning updates easier.

The energy function of the convRBM with Gaussian visible units which we use in our work is defined as

$$\begin{aligned} E_\theta(\mathbf{v},\mathbf{h}) = &- \left(\sum_k \sum_{B_i,B_j} \sum_{i',j' \in B_{B_i,B_j}} h^k_{i',j'} \cdot I\left(h^k_{i',j'} \right) \right)^{\frac{1}{D_h}} \\ &+ \frac{1}{2D_v\sigma^2} \sum_{ch} \sum_{r,s} \left(v_{r,s,ch} - c_{ch} \right)^2, \end{aligned} \tag{3.2}$$

subject to

$$\sum_{i',j' \in B_{B_i,B_j}} h^k_{i',j'} \leq 1, \quad \forall B_i, B_j, k, \tag{3.3}$$

where we used the notation

$$I\left(h^k_{i',j'}\right) = \frac{1}{\sigma^2}\left(\tilde{\mathbf{W}}^k *_v \mathbf{v}\right)_{i',j'} + b_k. \tag{3.4}$$

We use D_v to denote the dimensionality (area) of the input image $\mathbf{v}$, i.e. $D_v = v_{height} \cdot v_{width}$. Similarly, D_h is the dimensionality of the feature maps $\mathbf{h}^k$, $D_h = h_{height} \cdot h_{width}$, which depends on the size of the filters, i.e. $h_{height} = v_{height} - w_{height} + 1$ and $h_{width} = v_{width} - w_{width} + 1$. Including the dimensions of the input in the energy function allows us to train our model on differently sized images. The hidden feature maps $\mathbf{h}^k$ are split into a number of non-overlapping blocks B_{B_i,B_j} of size $C \times C$ where we refer to C as the pooling factor. We use $*_v$ to denote valid convolution and $\tilde{\mathbf{W}}$ to denote horizontal and vertical flipping of the filter $\mathbf{W}$. Note that the fact that $\mathbf{v}$ and $\mathbf{W}^k$ can have multiple channels is not stated explicitly. The constraint (Equation 3.3) is added as a way of including the probabilistic max-pooling in the energy function, which allows maximally one hidden unit to be active in every block B_{B_i,B_j}, as originally defined in [57]. This constraint is in practice satisfied by sampling from a multinomial distribution when sampling the hidden units. Having the energy function defined as in Equation 3.2 allows us to derive the learning updates by direct differentiation of the objective function, without adding any *ad hoc* normalizations as reported in other works (Equations 17, 18 and 19 in [58]), where the updates are divided by the dimensionality of the hidden or the visible layers.

3.1.2 Probabilistic semantics

The probabilistic semantics of the convolutional RBM are defined as follows. The probability that the *RBM* assigns to a configuration pair of the visible and hidden

units is

$$p_\theta(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E_\theta(\mathbf{v},\mathbf{h})}, \tag{3.5}$$

where Z is referred to as the *partition function* defined by summing over all possible pairs of the visible and hidden units:

$$Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E_\theta(\mathbf{v},\mathbf{h})}. \tag{3.6}$$

The probability of a single hidden unit $h^k_{i',j'}$ being on, given the input $\mathbf{v}$ is

$$p_\theta\left(h^k_{i',j'} = 1|\mathbf{v}\right) = \frac{e^{I\left(h^k_{i',j'}\right)}}{1 + \sum\limits_{i',j' \in B_{B_i,B_j}} e^{I\left(h^k_{i',j'}\right)}}. \tag{3.7}$$

This is also referred to as the activation of the hidden unit $h^k_{i',j'}$. The units in the pooling layer are sampled using the probabilistic max-pooling method, where a unit $p^k_{B_i,B_j}$ will be turned on if any of the hidden units in the block B_{B_i,B_j} is on. We can write the complement of its activation as:

$$p_\theta\left(p^k_{B_i,B_j} = 0|\mathbf{v}\right) = 1 - \sum_{i',j' \in B_{B_i,B_j}} p_\theta\left(h^k_{i',j'} = 1|\mathbf{v}\right). \tag{3.8}$$

The probability of sampling a single pixel in the input image $\mathbf{v}$ at position i', j' and channel ch, given the hidden units $\mathbf{h}^k$ is given by

$$p_\theta(v_{i',j',ch}|\mathbf{h}) = \mathcal{N}\left(\sum_k (\mathbf{W}^k *_f \mathbf{h}^k)_{i',j'} + c_{ch}, \sigma^2\right), \tag{3.9}$$

where $*_f$ denotes full convolution. The probability that is assigned to a specific sample $\mathbf{v}$ is calculated by summing over all possible hidden vectors:

$$p_\theta(\mathbf{v}) = \frac{1}{Z}\sum_{\mathbf{h}} e^{-E_\theta(\mathbf{v},\mathbf{h})}, \tag{3.10}$$

which can also be written in terms of *free energy* [10, 36] as

$$p_\theta(\mathbf{v}) = \frac{1}{Z'}e^{-F_\theta(\mathbf{v})}, \tag{3.11}$$

where

$$Z' = \sum_{\mathbf{v}'} e^{-F_\theta(\mathbf{v}')}. \tag{3.12}$$

If we use the energy function as defined in Equation 3.2 and rewrite Equation 3.10 in form of Equation 3.11 we see that the convRBM with Gaussian visible units has the free energy defined as:

$$\begin{aligned} F_\theta(\mathbf{v}) = &\frac{1}{2D_v\sigma^2}\sum_{ch}\sum_{r,s}(v_{r,s,ch} - c_{ch})^2 \\ &- \frac{1}{D_h}\sum_{k}\sum_{B_i,B_j}\log\left(1 + \sum_{i',j'\in B_{B_i,B_j}} e^{I\left(h^k_{i',j'}\right)}\right), \end{aligned} \tag{3.13}$$

where we use $I\left(h^k_{i',j'}\right)$ as defined in Equation 3.4. Details of how Equation 3.13 was derived can be found in Appendix A.

3.1.3 Training using contrastive divergence

In order to learn the parameters of a Convolutional RBM, we can define the maximum likelihood learning objective as:

$$f_{ML}(\theta) = \sum_{\mathbf{v}\in\mathbf{X}} p_e(\mathbf{v})\log p_\theta(\mathbf{v}) = \frac{1}{N}\sum_{n=1}^{N}\log p_\theta(\mathbf{v}_n), \tag{3.14}$$

where we used $p_e(\mathbf{v})$ when referring to the empirical distribution of the data in the training set $\mathbf{X}$, and $p_\theta(\mathbf{v})$ when referring to the model distribution, defined in Equation 3.11. Maximizing $f_{ML}(\theta)$ is equivalent to minimizing the *Kullback-Leibler divergence*, defined as:

$$D_{KL}(p_e||p_\theta) = \sum_{\mathbf{x}\in\mathbf{X}} p_e(\mathbf{x}) \log \frac{p_e(\mathbf{x})}{p_\theta(\mathbf{x})}, \tag{3.15}$$

which equals to zero if p_e and p_θ are equal. Maximizing the log-likelihood is the same as minimizing the negative log-likelihood, and we can write the gradient of $-\log p_\theta(\mathbf{v})$ as

$$\begin{aligned} -\frac{\partial \log p_\theta(\mathbf{v})}{\partial \theta} &= -\frac{\partial}{\partial \theta}\left(-F_\theta(\mathbf{v}) - log(Z)\right) = \\ &= \frac{\partial F_\theta(\mathbf{v})}{\partial \theta} + \frac{1}{Z}\frac{\partial Z}{\partial \theta} = \\ &= \frac{\partial F_\theta(\mathbf{v})}{\partial \theta} + \frac{1}{Z}\sum_{\tilde{\mathbf{v}}} \frac{\partial}{\partial \theta}\left(e^{-F_\theta(\tilde{\mathbf{v}})}\right) = \\ &= \frac{\partial F_\theta(\mathbf{v})}{\partial \theta} - \sum_{\tilde{\mathbf{v}}} p_\theta(\tilde{\mathbf{v}}) \frac{\partial F_\theta(\tilde{\mathbf{v}})}{\partial \theta}. \end{aligned} \tag{3.16}$$

Calculating $p_\theta(\tilde{\mathbf{v}})$ in the equation above is intractable because it involves the calculation of the partitioning function Z, so the second term has to be approximated. We can do this by writing [62]:

$$-\frac{\partial \log p(\mathbf{v})}{\partial \theta} \approx \frac{\partial F_\theta(\mathbf{v})}{\partial \theta} - \frac{1}{S}\sum_{s=1}^{S} \frac{\partial F_\theta(\tilde{\mathbf{v}}_s)}{\partial \theta}, \tag{3.17}$$

where $\tilde{\mathbf{v}}_s$ is a *negative particle*, a sample generated by sampling from the model using a *Markov Chain Monte Carlo* method, for example *block Gibbs sampling*. In the case of maximum likelihood optimization, generating the sample $\tilde{\mathbf{v}}_s$ requires running the Gibbs sampling chain until an equilibrium state is reached. Instead of running the Gibbs chain until it reaches equilibrium, an approximated approach can be used that runs the chain only for k steps, called *Contrastive Divergence*. Contrastive Divergence [35], [37] defines the objective function as the difference between two Kullback-Leibler divergences:

$$f_{CD}^k(\theta) = D_{KL}(p_e||p_\theta) - D_{KL}(q_\theta^k||p_\theta). \tag{3.18}$$

In the function defined like that, the terms that involve running the Gibbs chain infinitely many times cancel out, which makes Contrastive Divergence a method that can be used in practice. There is still a term in the objective function which is hard to compute, but as was shown empirically in [35], it can be safely ignored. This leads us to the approximated version of the gradient of the objective function:

$$\frac{\partial}{\partial\theta} f_{CD}^{k}(\theta) \approx -\frac{1}{N}\sum_{n=1}^{N}\left(\frac{\partial F_\theta(\mathbf{v})}{\partial\theta} - \frac{\partial F_\theta(\tilde{\mathbf{v}})}{\partial\theta}\right). \tag{3.19}$$

The gradient consists of two terms, called the *positive* and *negative phase*. The positive phase is there to increase the probability of the data given during training, while the negative phase decreases the probabilities of the data that is generated by the model [10]. In case of using k-step Contrastive Divergence, the negative particle $\tilde{\mathbf{v}}$ is generated by running the Gibbs chain for k steps. Finally, we define the approximate contrastive

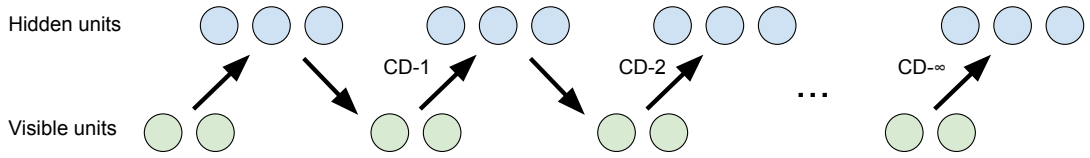


Figure 3.2: Illustration of Contrastive Divergence learning with k steps of sampling.

divergence objective function for the convolutional RBM as:

$$\mathcal{L}_{CD_k} = F_\theta(\mathbf{v}) - F_\theta(\tilde{\mathbf{v}}), \tag{3.20}$$

where $\tilde{\mathbf{v}}$ is generated by sampling from the model by running *block Gibbs sampling* for k (usually 1) steps. When sampling the negative particle from the model, we use the expected value of Equation 3.9.

In order to keep the weights of the learned filters small and to prevent overfitting, we regularize the cost function by adding an L2 norm regularization term for the weights:

$$R_{L2} = ||\mathbf{W}||_2^2 = \sum_{k,ch,i,j} w_{k,ch,i,j}^2. \tag{3.21}$$

The second regularization that we use is the sparsity regularization which tries to keep the mean activations of the hidden layer close to a target value. This helps to prevent the model of learning trivial solutions as it is overcomplete. We add this regularization as an additional term in the updates of the hidden biases b_k, as it was done in [58].

Learning updates

The derivatives of the free energy function as defined in Equation 3.13 with respect to all of the model parameters are calculated in the following way. Details on how these were derived can be found in Section 3.1.3. With respect to each element $w^k_{r,s}$ of the filter $\mathbf{W}^k$ we can write:

$$\frac{\partial F_\theta(\mathbf{v})}{\partial w^k_{r,s}} = -\frac{1}{D_h \sigma^2} \sum_{B_i, B_j} \sum_{\substack{i', j' \in \\ B_{B_i, B_j}}} p_\theta \left(h^k_{i',j'} = 1 | \mathbf{v} \right) v_{i'+r, j'+s}, \tag{3.22}$$

with respect to the hidden bias b^k:

$$\frac{\partial F_\theta(\mathbf{v})}{\partial b^k} = -\frac{1}{D_h} \sum_{B_i, B_j} \sum_{i', j' \in B_{B_i, B_j}} p_\theta \left(h^k_{i',j'} = 1 | \mathbf{v} \right), \tag{3.23}$$

and with respect to the visible bias in the ch-th channel c_{ch}:

$$\frac{\partial F_\theta(\mathbf{v})}{\partial c_{ch}} = -\frac{1}{\sigma^2 D_v} \sum_{r,s} v_{r,s,ch} - c_{ch}. \tag{3.24}$$

The updates for the objective function (Equation 3.20) follow directly from above and are equivalent to the updates in [58] when $\sigma = 1$ and if the term $\frac{1}{2D_v\sigma^2} \sum_{ch} \sum_{r,s} \left(v_{r,s,ch} - c_{ch}\right)^2$ is replaced by $\frac{1}{D_v\sigma^2} \left(- \sum_{ch} c_{ch} \sum_{r,s} v_{r,s,ch} + \frac{1}{2} \sum_{r,s,ch} v^2_{r,s,ch} \right)$ in Equation 3.2. If we include the regularization from Equation 3.21, we get

$$\Delta\theta = \frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial F_\theta(\mathbf{v})}{\partial \theta} - \frac{\partial F_\theta(\tilde{\mathbf{v}})}{\partial \theta} + \lambda_{L2} \frac{\partial R_{L2}}{\partial \theta}. \tag{3.25}$$

In order to include the sparsity regularization we follow [57] and add a term to the updates of the hidden biases b^k:

$$\begin{aligned} \Delta b^k_{sparsity} &= \lambda_{sparsity} \left(\frac{1}{D_h} \sum_{i',j'} p_\theta \left(h^k_{i',j'} = 1 | \mathbf{v} \right) - p_{target} \right), \\ \Delta b^k &:= \Delta b^k + \Delta b^k_{sparsity}, \end{aligned} \tag{3.26}$$

where p_{target} is the target sparsity and $\lambda_{sparsity}$ the sparsity gain coefficient that controls the amount of regularization. The training is done using mini-batch gradient descent with momentum. Every parameter θ at iteration $t+1$ is updated using the following rule:

$$\begin{aligned} v^{t+1}_\theta &= \gamma v^t_\theta + \mu \frac{1}{m} \sum_k^m \Delta\theta(\mathbf{v}_k) \\ \theta^{t+1} &= \theta^t - v^{t+1}_\theta, \end{aligned} \tag{3.27}$$

where γ denotes the momentum, μ is the learning rate and m is the size of the mini-batch.

3.2 Experiments and results

In this section we describe the experiments using the methods described in the earlier sections on the standard UCF-101 action recognition dataset.

3.2.1 Implementation details

The implementation of the model and all of the experiments described in this chapter were done in Python using Theano [100].

To show the speedup achieved by implementing the convRBM model using the Theano library and the speedup of training the convRBM on a GPU instead on a CPU

we compare it to the baseline Matlab implementation of [57] taken from the author's website[1]. We set up an experiment in which we train a single layer convRBM consisting of 24 filters of size 10×10px on 70×70px patches extracted from 10 natural images provided as demo data with the baseline code. The minibatch size was set to 1 in the experiment in order to be comparable to the results of the baseline implementation. Running the training for 500 epochs took on average around 21 minutes when using the Matlab implementation[2]. Running the same experiment on the same CPU, but using our Theano implementation took 115 seconds on average. Finally, running the same experiment on a Nvidia GeForce GTX Titan X (Maxwell) GPU took on average 16.5 seconds. This shows that using our implementation of the convRBM which can be run on a GPU resulted in a 76 times lower running time than when the baseline CPU implementation was used.

3.2.2 UCF-101 action recognition dataset

The UCF-101 dataset introduced by [93] is one of the most challenging action recognition datasets today, consisting of 13320 realistic action videos collected from YouTube belonging to 101 different action categories. More details on the UCF-101 dataset can be found in Appendix B. The classification performance on this dataset is reported as the mean accuracy over three provided train/test splits. In this chapter we report the accuracy on the first split only.

3.2.3 Training of the model

The first train/test split of the UCF-101 dataset contains 9537 train videos with around 1 700 000 frames in total. In this work we train the model only on static frames. We run the convRBM training and the feature extraction on a Tesla K-40 GPU.

[1] Baseline implementation: http://web.eecs.umich.edu/~honglak/softwares/crbm_demo_r2.tgz
[2] The experiment was run on a Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz.

Preprocessing

All the frames are converted into grayscale, each pixel in all of the frames in the training set is normalized to zero mean and unit variance and all the frames are whitened using a low-pass filter. As some of the videos in the dataset contain black borders on the top and bottom of the video, we crop all of the videos from the original 320×240 px to 320×180 px. During training each frame is flipped horizontally with 50% probability.

Training details

We train the convRBMs as described in Section 3.1.3. The label information provided in the training set was ignored during the training of the convRBMs. The first layer convRBMs we trained consist of either 32 or 64 filters of size 5×5 or 3×3 pixels. We train the second convRBM on either the pooling or the hidden layer activations of the first convRBM. If we use pooling we set the pooling factor C to 2. Second layer convRBMs contain 64 filters with size 3×3 pixels and the third layer convRBMs contain 128 filters with size 3×3 pixels. We set the mini-batch size to 10, λ_{L2} to 0.01 and σ to 0.2 in all of the experiments. The sparsity gain $\lambda_{sparsity}$ was set to 1000 when training the first convRBM and to 10 when training other layers. The learning rate was set to 0.005 when using 5×5 filters, 0.003 when training second layer convRBMs and 0.002 for training all the convRBMs reported in the last row in Table 3.1. The sparsity targets used are shown in Table 3.1. The momentum is set to 0.5 in the start and later increased to 0.9. All the training was done on a random subset of 100 000 - 200 000 frames from the training set. The model from the last row in Table 3.1 was trained on around 30 000 frames.

3.2.4 Video representation

In order to be able to classify videos of different time durations we first need to find a video representation of a fixed length that can be used as input to a classifier such as an SVM. To achieve this we will follow a procedure similar to the procedure described

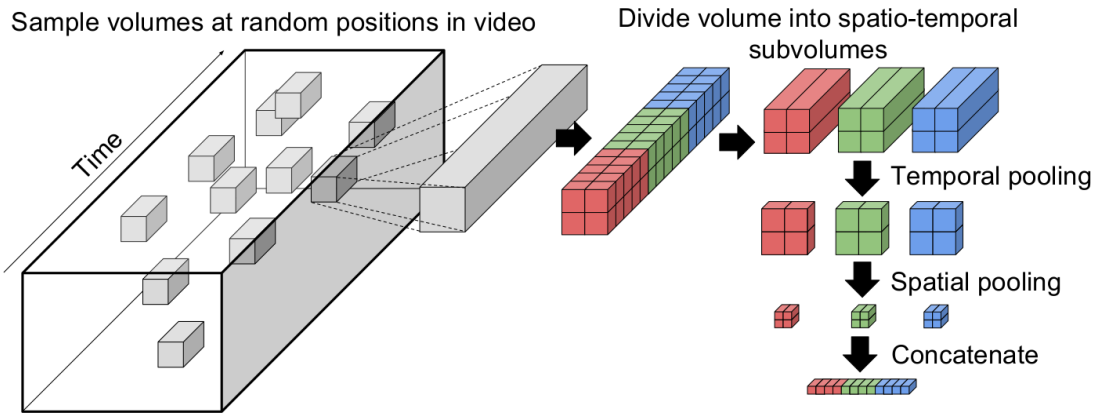


Figure 3.3: The used video representation. The concatenated vectors are used to train a GMM, from which a Fisher vector descriptor is extracted.

in [105].

In our experiments we follow the reasoning that if the features extracted from volumes at random positions in videos result in some performance, extracting the same features from volumes around trajectories should result in a higher performance. We do our experiments on representations extracted from volumes located at random positions. Each extracted volume contains 15 subframes of size 32×32 px. We use the trained convolutional DBN to extract the representations for every frame in the volume by doing approximate inference, using only one bottom-up pass in the network. We divide the resulting representation into a fixed number of subvolumes, e.g. into 2×2 spatial and 3 temporal regions, as it was done in [105]. After that, we pool the representation of each region using mean pooling into the length of a single feature. This way we can achieve some temporal and spatial invariance. In case we had 32 filters in the first convolutional RBM and we used the activations of the pool layer as the features, the final dimension of the features extracted from a single volume would be $2 \times 2 \times 3 \times 32 = 384$. An illustration of how we do the feature extraction from videos can be seen in Figure 3.3.

After we extracted the features from each of the volumes, we randomly sample a number of them and use them to perform PCA in order to reduce the feature dimen-

sionality and decorrelate the features. We then train a Gaussian mixture model on those features and finally, use the GMM to extract the Fisher vector representation of the whole video. The Fisher vectors are normalized using the $L2$ and power normalizations from [85]. A detailed algorithm for extracting Fisher vectors can be found in [85]. Until this point, we have not used any label information provided in the training set.

3.2.5 Classification results using stacks of convolutional RBMs

In this subsection we report the results of action classification achieved by the methods described above on the first split of the UCF-101 dataset. All of the classification was done using a linear SVM with C set to 100, as in the work reported in [105]. The video representation that we chose for our experiments is the one described in Section 3.2.4. During training we extract 1000 volumes from each of the 9537 training videos. Each volume consists of 15 adjacent frames with dimensions 32×32 pixels in the input video. In case when the first convRBM contains 32 filters of size 5×5, with a pooling factor $C = 2$, this would map to a 28×28 pixel region in the first hidden layer and a 14×14 pixel region in the first pool layer. Similarly, if we have 64 filters of size 3×3 in the next convRBM, the corresponding region in the second pool layer would be 6×6 pixels big, if we used the pooling layer of the first convRBM when training the second convRBM. All the volumes in our experiments are extracted from random positions. The positions of where the volumes are extracted in different experiments are always the same so we can directly compare the results.

After extracting the volumes, each of them is partitioned into $2 \times 2 \times 3$ smaller subvolumes. We extract the features we chose to use in a particular experiment from every subframe in the subvolume, and pool them using mean pooling. Using this method, the dimensionality of each volume is then $2 \times 2 \times 3 \times D$, where D is the dimensionality of a single feature, e.g. 8 in the case of HOG.

After calculating the representation of every extracted volume, we randomly chose 30 of the volumes per video to get 286 110 volumes that we then use to apply PCA on for dimensionality reduction and to decorrelate the data. We chose this number to be similar to the 256 000 used in [107]. We decide to keep the number of principal components that explain 0.95 of the data variance, rounded up to the closest hundred. In case of HOG, we chose to reduce the dimensionality to 48 to follow the same procedure as reported in [107]. Again following the same work, we use the same volumes we picked to train a Gaussian mixture model with $K = 256$ Gaussians, using the expectation maximization algorithm. After the GMM has been trained, we can use it to extract the Fisher vector descriptors for each of the videos. Each FV's dimensionality will be $K \times (2 \times D + 1)$, where K is the number of Gaussians in the GMM and D is the dimensionality of the features used. Finally, we train the linear SVM classifier with $C = 100$ using the extracted Fisher vector representation of each video in the training set as input. This is the only part of our proposed architecture that is trained in a supervised way, requiring the knowledge of action class labels assigned to each video in the training set. All the other parts of the architecture that are used for extracting the representation of a video were trained in an unsupervised way.

We report the experiments and the achieved classification accuracy on the first train/test split of the UCF-101 dataset in Table 3.1. These experiments were done using only a single type of features per video. We report results we got using the standard HOG features and features extracted from different layers in the convDBN architecture.

3.3 Discussion and conclusion

We show the performance of features extracted at different layers of our model in Table 3.1. The features were extracted at 1000 random positions in each video (same

Table 3.1: Classification accuracy achieved on the UCF-101 dataset (1st training/testing split).

Features used	**ConvRBM trained on**	**Sparsity target**	**Filter dim.**	**Feature dim.**	**After PCA**	**FV dim.**	**Acc.**
HOG	-	-	-	96	48	24832	50.75%
Trajectories	-	-	-	28	-	14592	43.51%
1st pool layer	Input	-	$32 \times 5 \times 5$	384	100	51456	45.36%
1st pool layer	Input	0.015	$32 \times 5 \times 5$	384	100	51456	50.86%
1st hidden layer	Input	0.008	$32 \times 5 \times 5$	384	100	51456	36.66%
1st pool layer	Input	0.008	$32 \times 5 \times 5$	384	100	51456	48.40%
2nd pool layer	1st pool	0.01	$64 \times 3 \times 3$	768	200	102656	45.25%
1st pool layer	Input	0.02	$32 \times 5 \times 5$	384	100	51456	52.95%
2nd pool layer	1st pool	0.02	$64 \times 3 \times 3$	768	200	102656	49.77%
2nd pool layer	1st pool	0.03	$64 \times 3 \times 3$	768	200	102656	49.67%
1st pool layer	Input	0.03	$32 \times 5 \times 5$	384	100	51456	53.19%
2nd pool layer	1st pool	0.02	$64 \times 3 \times 3$	768	200	102656	49.91%
1st pool layer	Input	0.03	$64 \times 5 \times 5$	768	100	51456	55.06%
1st pool layer	Input	0.015	$64 \times 5 \times 5$	768	100	51456	52.47%
1st hidden layer	Input	0.03	$64 \times 3 \times 3$	768	100	51456	41.50%
1st pool layer	Input	0.03	$64 \times 3 \times 3$	768	100	51456	50.36%
2nd pool layer	1st hidden	0.017	$64 \times 3 \times 3$	768	100	51456	51.73%
2nd pool layer	1st hidden	0.04	$64 \times 3 \times 3$	768	100	51456	54.22%
3rd pool layer	2nd pool	0.04	$128 \times 3 \times 3$	1536	200	102656	51.28%
3rd pool layer	2nd hidden	0.04	$128 \times 3 \times 3$	1536	200	102656	51.94%

Table 3.2: The state of the art results reported on the UCF-101 dataset. Last two columns show whether the used features are static or dynamic and if they were trained supervised, unsupervised or are handcrafted.

#	**Method (year)**	**Accuracy**	**Static/Dyn.**	**Sup/Unsup/Des**
1	Pooling Convolutional Layers [117] (2015)	93.78%	Stat. + Dyn.	Supervised
2	Trajectory-Pooled Deep-Convolutional Descriptors [111] (2015)	91.5%	Stat. + Dyn.	Supervised
3	Two-Stream CNN [89] (2014)	88.0%	Stat. + Dyn.	Supervised
4	Improved Dense Trajectories + HOG + HOF + MBH [107] (2013) [108]	85.9%	Stat. + Dyn.	Handcrafted
5	Improved Dense Trajectories + MBH [107] (2013) as reported in [108]	82.1%	Dynamic	Handcrafted
6	Improved Dense Trajectories + HOF [107] (2013) as reported in [108]	78.3%	Dynamic	Handcrafted
7	Improved Dense Trajectories + HOG [107] (2013) as reported in [108]	74.6%	Static	Handcrafted
8	Single Frame Optical Flow CNN [89] (2014)	73.9%	Dynamic	Supervised
9	Single Frame Image CNN [89] (2014)	73.0%	Static	Supervised
10	Slow Fusion Network [46] (2014)	65.4%	Static	Supervised
11	Dense Trajectory + MBH, encoded with BOW [105] (2011) as reported in [8]	62.93%	Dynamic	Handcrafted
12	**Ours**	**55.06%**	**Static**	**Unsupervised**
13	Dense Trajectory + HOF, encoded with BOW [105] (2011) as reported in [8]	51.10%	Dynamic	Handcrafted
14	Dense Trajectory, encoded with BOW [105] (2011) as reported in [8]	49.88%	Static	Handcrafted
15	Dense Trajectory + HOG, encoded with BOW [105] (2011) as reported in [8]	46.59%	Static	Handcrafted

throughout the experiments) and encoded using the procedure described in Section 3.2.4. The model was trained on static frames and we can compare its performance with the performance of the popular handcrafted HOG features. We show that the features extracted by the convolutional DBN result in a higher classification accuracy than HOG features, even when using only one convolutional RBM. We also show how

the sparsity regularization affects the performance. We can see that setting the sparsity target too low resulted in a degraded performance. Also, we trained the higher layer convolutional RBMs either on the hidden layer or the pooling layer activations of the previous convRBM. Training on the hidden layer seems to be a better choice.

3.3.1 State of the art results

We list some of the relevant and recent state of the art results achieved on the UCF-101 dataset in Table 3.2. The methods that achieve the best performance make use of the two-stream network idea introduced in [89], where one network is used to extract a representation from static frames and a second one that uses optical flow as input. This includes the works of [117] and [111]. The work of [46] examines different kinds of extending the CNN into time domain. In [105] and [107] a dense grid of points is tracked using optical flow to form trajectories. The trajectories in [107] are further improved by ignoring camera motion by estimating the homography between adjacent frames. A number of different handcrafted features is extracted along the trajectories which are then encoded using Fisher vectors [85] (rows 4-7 in Table 3.2) or using the BOW model (rows 11, 13, 14, 15).

The works that are more closely related to ours, and allow for a meaningful comparison, are works that use static features (marked with *static* in Column 4 in Table 3.2) that either learn their features in an unsupervised way, or use handcrafted features (marked in column 5 in Table 3.2). From those, our method outperforms [105] that use static HOG features and a BOW model, but is worse than [107] that use improved dense trajectories and a Fisher vector representation, possibly because of the use of improved dense trajectories and a video stabilization scheme. We note that we extracted our features at a fixed number of random positions in the videos, whereas in [107] or [111] the features are extracted densely and around trajectories - also those methods use as features the displacements along the trajectories. As we have shown in Table 3.1, by using our features with random sampling in the spatio-temporal video

volume, we outperform considerably the directly comparable method that does random sampling using HOG descriptors. We can reasonably expect that extracting our features along dense trajectories is likely to result in a better performance than HOG features extracted along trajectories.

Finally, let us note, that the models we used in this chapter to extract the video representations were trained in an unsupervised way, in contrast to the state of the art methods where the models are trained supervised. Adding supervision to our training procedure would prevent the model from trying to learn intraclass variations and focus it more on learning discriminative features. This should also increase the performance of the learned representations - we aim to investigate this in the following chapter.

3.3.2 Conclusion

In this chapter we have shown how features extracted from a convolutional deep belief network trained in an unsupervised way on static video frames can be used for the problem of human action recognition. We modified the energy function of the convolutional RBM in such a way that the learning updates defined in the literature follow directly from differentiation of the objective function, without applying any additional normalizations. Having an objective function defined in this way allowed us to use the Theano library which supports automatic differentiation and provides a way to run experiments on a GPU instead of a CPU. Moving the computation to the GPU resulted in a 76x speedup compared to the baseline CPU implementation of convolutional RBMs written in Matlab. Speeding up the training procedure is an important achievement as training deep learning architectures usually requires processing large amounts of data which is not practical when only a CPU implementation is available.

Although the performance of the proposed method and architecture applied to the problem of action recognition surpassed the performance of approaches that use handcrafted features such as HOG or HOF encoded with a BOW model, it is still way

behind the state of the art results achieved by deeper architectures trained using supervised learning methods. Increasing the depth of our architecture by training additional convolutional RBM layers on top of the first layer did not result in an expected performance improvement reported by other works in the literature. We argue that this is because we did not find a meaningful way of picking the most suitable hyperparameters for training of the higher layer convolutional RBMs. Choosing the right set of training hyperparameters requires further research.

The other main cause of the inferior performance of the architectures used in this chapter when compared to the state of the art architectures applied for action recognition is the fact that other models are trained in a supervised manner. As mentioned before, having supervision added to the training procedure should allow the model to learn more discriminative features, instead of learning features that try explaining the intraclass variations of the training data. With the goal of pushing the performance of our models more towards the state of the art, we are going to investigate increasing the architecture's depth and switching to supervised training methods, which will be one of the topics of the following chapter.

Chapter 4

End to end trainable convolutional Fisher vector network for action recognition

Contents

In this chapter we propose a novel neural network architecture for the problem of human action recognition in videos. The proposed architecture expresses the processing steps of classical Fisher vector approaches, that is feature extraction, dimensionality reduction by principal component analysis (PCA) projection, Gaussian mixture model (GMM) fitting and Fisher vector descriptor extraction, as network layers. By contrast to other methods where these steps are performed consecutively and the corresponding parameters are learned in an unsupervised manner - as it was done in the previous chapter where we used convolutional restricted Boltzmann machines for the feature extraction step, having all the steps defined as a single neural network allows us to refine the whole model discriminatively in an end to end fashion. Furthermore, we

show that the proposed architecture can be used as a replacement for the fully connected layers in popular convolutional networks achieving a comparable classification performance, or even significantly improving the performance while reducing the total number of trainable parameters by a factor of 5. We show that our method achieves significant improvements in comparison to the classical chain.

In this chapter we describe a method that expresses all the different steps of the action recognition using FV, as layers in a neural network. The layers are initialized by unsupervised training in a layer by layer manner, and are subsequently refined in an end-to-end training. Our network with the layers initially trained offline directly corresponds to the standard Fisher vector pipeline. However, having the pipeline defined as a network, allows for discriminative finetuning which we will show brings significant improvements over the baseline pipeline. The proposed architecture results in spatio-temporal descriptors at intermediate levels of the architecture, calculated by local aggregation in a spatio-temporal structure of frame-level descriptors. More specifically, the main contributions of this work are the following:

- We describe a novel neural network architecture for action recognition which includes two new types of layers; the Gaussian mixture model layer and the Fisher vector descriptor layer. Combining these layers with other standard ones into a single deep neural network gives us a way of jointly finetuning the parameters of the whole architecture with respect to a chosen discriminative cost, using the standard backpropagation algorithm. We show that adding supervision at every stage of the network improves the discriminative power of the extracted Fisher vector descriptor compared to the standard version of the descriptor which is extracted in an unsupervised manner.

- Analogous to convolutional neural networks, where the same operation is applied at different locations of the input tensor, our network offers a natural way of extracting the Fisher vector descriptors densely from a given input video, both

in space and in time. This also allows us to easily extract the descriptor only from selected parts of the video, providing a straightforward way of implementing other architectures, such as spatial pyramids.

- We show that the proposed architecture can be used as a replacement for the fully connected layers in popular convolutional networks such as the VGG-16 network, achieving a comparable classification performance while reducing the total number of trainable parameters by a factor of 5.

The main idea behind FVs is to encode a set of local descriptors extracted from a sample (i.e. an image or a video) as a vector of deviations from the parameters of a generative model (usually a Gaussian mixture model) fitted to the descriptors extracted on the training set. Although FVs are good global descriptors on their own, there are shortcomings in the way they are extracted. Namely, the GMM used for encoding is learnt in an unsupervised way without receiving any additional information about the task at hand. This results in descriptors that are not tailored for a discriminative task as the GMM also learns to model the intra-class variations of the training data something that is not relevant for classification problems.

4.1 Discriminative convolutional Fisher vector network

In this section we describe the proposed architecture of our discriminative convolutional Fisher vector network - an illustration is given in Figure 4.1. We start by describing each of the used layers in detail and give the final overview of the whole architecture in Subsection 4.1.7.

The architecture can be divided into six parts; the local feature extraction layers, the spatio-temporal pooling layer, the dimensionality reduction layer, the Gaussian mixture layer, the Fisher vector descriptor layer and the classification layer. Their descriptions follow in Subsections 4.1.1, 4.1.2, 4.1.3, 4.1.4, 4.1.5 and 4.1.6.

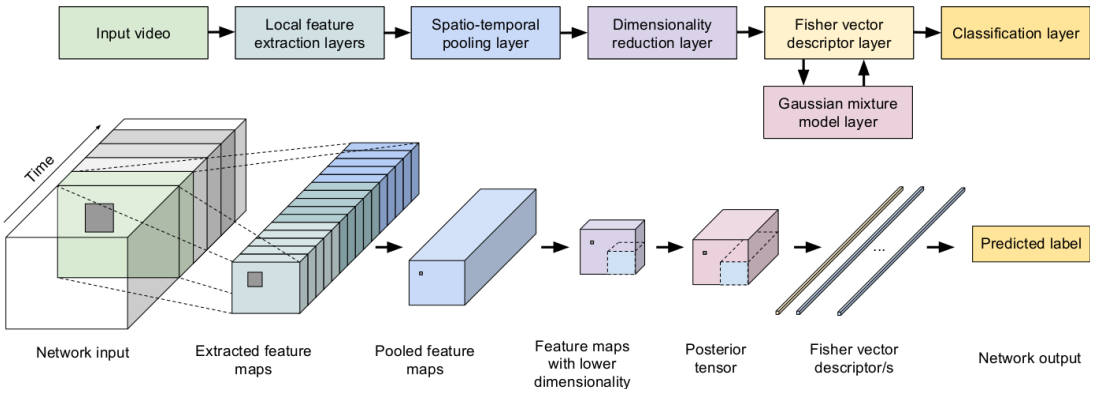


Figure 4.1: An illustration of the proposed architecture. The input to the network is a stack of t static frames (marked in green in the left of the figure) from a video which are passed through the feature extraction layers resulting in t feature maps of size $F_h \times F_w \times d$ where d is the number of channels. The feature maps are then pooled temporally and spatially which results in new feature maps of size $F'_h \times F'_w \times D$ whose dimensionality is reduced in the dimensionality reduction layer to $F'_h \times F'_w \times n_c$. The Fisher vector descriptor layer passes these feature maps to the GMM layer which gives a tensor of posteriors at its output. Using the posteriors and the input feature maps, the FV layer outputs the FV descriptor of the t frames of the video. Note that we can use different crops of the posterior tensor in order to calculate the FV descriptor of only a part of the network input (e.g. using the subtensor marked in blue in the posterior tensor would correspond to calculating the FV descriptor for only the bottom right corner of the input video). Finally the FV descriptors are normalized and fed into the classification layer where their scores are averaged and used to predict the label for the given input. In order to predict a label for the whole video of length L, we slide the network along the time axis with a stride of δ_T frames, updating the FV descriptor/s on the way. The classification step is the same as when predicting the label of a stack of t frames.

4.1.1 Local feature extraction layers

To do the first step of local feature extraction in our architecture, we tried using two different networks. We fed static video frames into a small network consisting of a single convolutional layer followed by a pooling layer. This part of the network was pretrained on static video frames using a convolutional restricted Boltzmann machine [57] as described in [76], that is Chapter 3, with the difference that we used local contrast normalization as a preprocessing step. To show that the feature extraction layers can be replaced by any larger and more complex network, we also used the VGG-

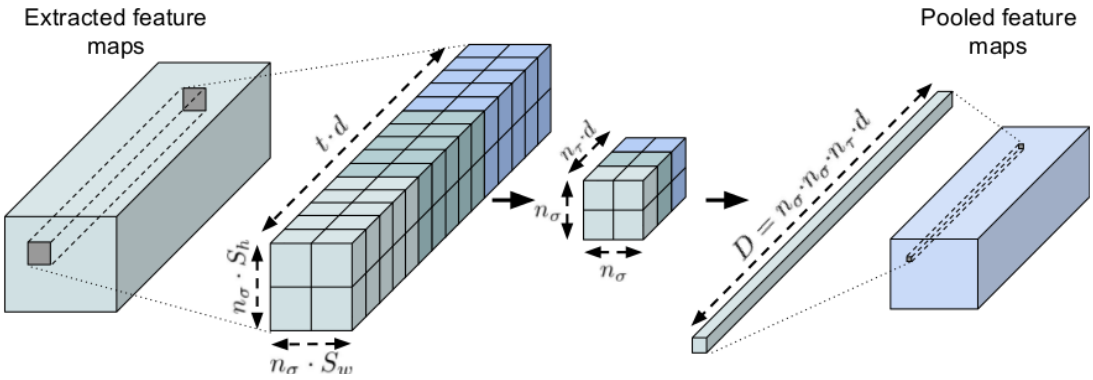


Figure 4.2: An illustration of how the spatio-temporal layer performs pooling on the extracted feature maps. Subtensors of size $n_\sigma \cdot S_h \times n_\sigma \cdot S_w \times t \cdot d$ are pooled into tensors of size $n_\sigma \times n_\sigma \times n_\tau \cdot d$ and resized into vectors of dimensionality D. This is repeated for all subtensors in the extracted feature maps, sliding horizontally and vertically with a stride of δ_S.

16 [90] network pretrained on the ImageNet dataset. Given L consecutive images, the output of the feature extraction layers is L feature maps of size $F_h \times F_w \times d$, where F_h denotes its height, F_w the width and d the number of its channels.

4.1.2 Spatio-temporal pooling layer

In order to include motion information from the input video, we want to combine feature maps extracted from multiple static frames into a more powerful representation. To do so, we follow the work of [105] where a spatio-temporal volume of features extracted from t frames is divided into $n_\sigma \times n_\sigma \times n_\tau$ subvolumes of size $S_h \times S_w \times t/n_\tau$ and then pooled temporally and spatially using mean pooling. The resulting representation is then resized into a vector of dimensionality $D = n_\sigma \cdot n_\sigma \cdot n_\tau \cdot d$, where d is the dimensionality of the local features extracted from the previous layer. This vector is then used as input to the following layer. In the case that the tensor at the input of this layer has bigger spatial dimensions than $n_\sigma \cdot S_h \times n_\sigma \cdot S_w$, the described pooling procedure is repeated for each $n_\sigma \cdot S_h \times n_\sigma \cdot S_w \times t \cdot d$ subtensor, moving through the tensor with a spatial stride δ_S as shown in Figure 4.2. Later we will describe how we deal with longer time periods. Given t feature maps of size $F_h \times F_w \times d$ at input, the

output of this layer is a tensor of size $F'_h \times F'_w \times D$, where $F'_h = (F_h - S_h)/\delta_S + 1$ and $F'_w = (F_w - S_w)/\delta_S + 1$.

4.1.3 Dimensionality reduction layer

In the standard Fisher vector pipeline the locally extracted features are decorrelated and their dimensionality is reduced by performing PCA. Assuming that the mean of the data μ_x and the principal axes P were found offline, the mapping from the original data to a lower-dimensional space can be written as:

$$\boldsymbol{x}'_t = (\boldsymbol{x}_t - \boldsymbol{\mu}_x)\boldsymbol{P}'. \tag{4.1}$$

The dimensionality of matrix $\boldsymbol{P}$ is $n_c \times D$, where n_c is the number of components and D is the dimensionality of original data. Note that we do not put any constraints on the matrix $\boldsymbol{P}$, so after backpropagating through the layer and updating its parameters the projection applied on the input data is not guaranteed to be orthogonal. Given a tensor of size $F'_h \times F'_w \times D$ at input, the output of this layer is a tensor of size $F'_h \times F'_w \times n_c$.

4.1.4 Gaussian mixture model layer

A Gaussian mixture model is defined as a weighted sum of K components [85]:

$$u_\lambda(\boldsymbol{x}) = \sum_{k=1}^{K} w_k u_k(\boldsymbol{x}), \tag{4.2}$$

where w_k is a component weight and $u_k(\boldsymbol{x})$ is a probability density function of the Gaussian distribution:

$$u_k(\boldsymbol{x}) = \frac{1}{(2\pi)^{\frac{D}{2}} |\boldsymbol{\Sigma}_k|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_k)' \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_k)\right). \tag{4.3}$$

Every GMM can be described by the parameter set $\lambda = \{w_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, k = 1, ..., K\}$, where w_k is the k-th component weight, $\boldsymbol{\mu}_k$ is its mean vector and $\boldsymbol{\Sigma}_k$ its covariance matrix. The mixture coefficients $\{w_k\}$ are constrained to be positive and to sum to

one, that is $w_k \geq 0$ and $\sum_{k=1}^{K} w_k = 1$ which can be easily enforced by using internal weights α_k and defining:

$$w_k = \frac{\exp(\alpha_k)}{\sum_l^K \exp(\alpha_l)}, \tag{4.4}$$

as it was done in [48]. For each sample $\boldsymbol{x}_t$, k posteriors describing the responsibility of each component for generating the sample can be found as:

$$\gamma_t(k) = \frac{w_k u_k(\boldsymbol{x}_t)}{\sum_l^K w_l u_l(\boldsymbol{x}_t)}. \tag{4.5}$$

When viewing the GMM as a neural network layer, we treat the sample $\boldsymbol{x}_t$ as the input to the layer and the posteriors $\{\gamma_t(k), k = 1, ..., K\}$ as its output. In the case when $\boldsymbol{x}_t$ is a n_c-dimensional vector, we can see that $u_k(\boldsymbol{x}_t)$ can be calculated by using subtraction, addition, multiplication, division and exponentiation which all are standard operations found in neural networks.

In a more general case, $\boldsymbol{x}_t$ can be treated as a 3-dimensional tensor, that can correspond to a feature map consisting of n_c channels of height F'_h and width F'_w. This tensor could, for example, be an image with 3 RGB channels, or any feature map outputted by a preceding layer. Following the same idea of convolution in convolutional layers, we can calculate both $u_k(\boldsymbol{x}_t)$ and $\{\gamma_t(k), k = 1, ..., K\}$ by performing the same operations we did in the case when $\boldsymbol{x}_t$ was a vector, repeating them for each of the $F'_h \cdot F'_w$ n_c-dimensional vectors in the tensor. This procedure will result in also a 3-dimensional tensor of size $F'_h \times F'_w \times K$. This is easily extended to the case of 4-dimensional tensors that usually appear in deep learning frameworks, with the first dimension corresponding to the number of samples in a minibatch.

4.1.5 Fisher vector descriptor layer

The Fisher vector descriptor is a global descriptor used to represent data by describing how the parameters of a generative model fitted on a distribution of local features should change in order to better model the local features extracted from the given data sample. An introduction to Fisher vectors and the underlying theory can be found in

[85]. In this subsection we show how the Fisher vector descriptors are calculated. If we define the statistics of the GMM as:

$$S_k^0 = \sum_{t=1}^{T} \gamma_t(k), \tag{4.6}$$

$$\boldsymbol{S}_k^1 = \sum_{t=1}^{T} \gamma_t(k)\boldsymbol{x}_t, \tag{4.7}$$

and

$$\boldsymbol{S}_k^2 = \sum_{t=1}^{T} \gamma_t(k)\boldsymbol{x}_t^2, \tag{4.8}$$

where $\gamma_t(k)$ is the k-th posterior from Equation 4.5, the parts of the Fisher vector corresponding to each of the parameters of the GMM can be calculated as follows:

$$\mathscr{G}_{w_k}^X = \left(S_k^0 - Tw_k\right)/\sqrt{w_k}, \tag{4.9}$$

where T represents the number of local descriptors,

$$\mathscr{G}_{\mu_k}^X = \left(\boldsymbol{S}_k^1 - \boldsymbol{\mu}_k S_k^0\right)/\left(\sqrt{w_k}\boldsymbol{\sigma}_k\right), \tag{4.10}$$

and

$$\mathscr{G}_{\sigma_k}^X = \left(\boldsymbol{S}_k^2 - 2\boldsymbol{\mu}_k\boldsymbol{S}_k^1 + (\boldsymbol{\mu}_k^2 - \boldsymbol{\sigma}_k^2)S_k^0\right)/\left(\sqrt{2w_k}\boldsymbol{\sigma}_k^2\right). \tag{4.11}$$

The resulting vectors are concatenated into a large vector:

$$\mathscr{G}_\lambda^X = \left(\mathscr{G}_{w_1}^X, \ldots, \mathscr{G}_{w_K}^X, \mathscr{G}_{\mu_1}^{X\prime} \ldots, \mathscr{G}_{\mu_K}^{X\prime}, \mathscr{G}_{\sigma_1}^{X\prime} \ldots, \mathscr{G}_{\sigma_K}^{X\prime}\right)', \tag{4.12}$$

which is the unnormalized version of the Fisher vector. The FV is normalized by applying two kinds of normalization; power normalization:

$$\left[\mathscr{G}_\lambda^X\right]_i \leftarrow sign\left(\left[\mathscr{G}_\lambda^X\right]_i\right)\sqrt{\left|\left[\mathscr{G}_\lambda^X\right]_i\right|}, \tag{4.13}$$

and L2 normalization:

$$\mathscr{G}_\lambda^X \leftarrow \mathscr{G}_\lambda^X / \sqrt{\mathscr{G}_\lambda^{X\prime}\mathscr{G}_\lambda^X}. \tag{4.14}$$

We can view the Fisher vector descriptor encoding as a network layer which contains an internal GMM layer and receives data, in the simplest case an n_c-dimensional vector,

$\boldsymbol{x}_t$ as input. The input data is passed to the internal GMM layer which gives the posteriors $\{\gamma_t(k), k = 1, \ldots, K\}$ at its output. The input data and the posteriors are then used to calculate the statistics from Equations 4.6, 4.7 and 4.8. We can see that the operations required in order to calculate both the unnormalized (Equation 4.12) and normalized (Equation 4.14) versions of the Fisher vector descriptor are all standard operations typically found in neural networks so the Fisher vector descriptor encoding can be easily expressed as a network layer. The dimensionality of the calculated FV is $d_{FV} = K \cdot (2n_c + 1)$.

Following the same reasoning as with the GMM layer when dealing with tensor data at input, the Fisher vector layer can also receive a tensor of size $F'_h \times F'_w \times n_c$ as its input. This tensor can be seen as a set of $F'_h \cdot F'_w$ n_c-dimensional vectors which we can then encode using the FV descriptor by the procedure described above. Note, that the FV encoding is performed by aggregating, across the first two modes of the tensor, the differential representations that are extracted along the fibers of the input tensor in the third mode. It is therefore trivial to extract the FV descriptor only from a selected subtensor of the input tensor. This allows us to use multiple crops of the input video during both the training and testing time in order to prevent overfitting and help improve generalization. This also allows us to create other architectures, such as the spatial pyramid [53].

4.1.6 Classification layer

Given a Fisher vector descriptor of a video or a part of a video we design the final layer of our network to output a prediction of the input video's class. To this end, we train m binary one-vs-all support vector machines as a classifier, where m is the number of classes. The cost that we use for optimizing the whole network is the squared SVM

hinge loss defined as:

$$C(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}) = \frac{\lambda}{2}||\boldsymbol{w}||^2 + \sum_{j}^{m} \max\left(0, 1 - y_j \cdot s_j\right)^2, \tag{4.15}$$

where $\boldsymbol{w}$ are the SVM weights, $\lambda = 2/(NC)$ with C being a regularization constant, $\boldsymbol{y}$ is the label encoded as a vector where all elements are -1 except for one that is 1, marking which class the input $\boldsymbol{x}$ belongs to, and with $\boldsymbol{s}$ being the SVM score, $\boldsymbol{s} = \boldsymbol{x}\boldsymbol{w}^T + \boldsymbol{b}$. We use y_j to denote the j-th element of the vector $\boldsymbol{y}$.

4.1.7 Fisher vector network for action recognition

The input to our network is a video represented as a stack of L consecutive static frames. In order to explain the pipeline of our architecture we will first limit the length of the input video to t, with $t \ll L$. Each of the t frames is passed through the local feature extraction layers which output t feature maps of size $F_h \times F_w \times d$, where d is the number of channels. These feature maps are then sent through the spatio-temporal pooling layer where they are pooled temporally and spatially as described in Subsection 4.1.2, resulting in a representation of size $F'_h \times F'_w \times D$. After passing this through the dimensionality reduction layer described in Subsection 4.1.3 the new representation is of a lower dimensionality, $F'_h \times F'_w \times n_c$. This is then passed into the Fisher vector descriptor layer described in Subsection 4.1.5 which internally uses the same input in the Gaussian mixture model layer from Subsection 4.1.4 to get a tensor of posteriors of size $F'_h \times F'_w \times K$. We can treat the input tensor as a set of $F'_h \times F'_w$ descriptors with n_c dimensions which also have corresponding $F'_h \times F'_w$ posteriors for each of the K components of the GMM layer. These are then used to calculate the needed statistics and get the unnormalized version of the Fisher vector. The FV can then be normalized and fed into the classification layer which gives the predicted label for the t frames of the original input.

For the unconstrained case when the whole video of L frames is to be classified

it is enough to notice that the unnormalized Fisher vector descriptor of a sequence of $2t$ frames is equal to the sum of the unnormalized FVs of the first t frames and the second t frames. Therefore, we can calculate the FV representation of the whole video by sliding our network along the time axis with a temporal stride of δ_T frames and summing the unnormalized FVs for each of the time segments. After normalizing the FV we can feed it into the classification layer to get the predicted label for the whole video. When several several crops are used, the resulting FV are fed to the classification layer and the corresponding outputs averaged.

The proposed neural network could also be viewed as a 3D "filter" with an internal representation which changes as the filter is "convolved" through a given video represented as a spatio-temporal volume. Once the filter passes through the whole video, the classification layer of the network uses the internal video representation, i.e. the Fisher vector descriptor, to give a prediction about the video's label. We show how our model could be used to predict labels for different parts of a video in Figure 4.3.

4.1.8 Number of trainable parameters

As our proposed network is a fully convolutional network, the number of its trainable parameters does not depend on the input's dimensions. Here we will summarize the total number of parameters learnt in each of our architecture's layers, excluding the local feature extraction layers as these can be replaced by an arbitrary network.

The PCA layer consists of two trainable parameters; a $D-$dimensional mean vector $\boldsymbol{\mu}_x$ and a $n_c \times D$ dimensional matrix of principal axes $\boldsymbol{P}$, where n_c denotes the number of components kept after applying the projection and D is the number of channels of the tensor returned from the spatio-temporal pooling layer. The FV layer consists of a GMM layer that contains K trainable sets of parameters w_k, μ_k and Σ_k, where w_k is a scalar, $\boldsymbol{\mu}_k$ is a n_c-dimensional vector and $\boldsymbol{\Sigma}_k$ is diagonal matrix containing n_c trainable parameters. The classification layer consists of a $m \times d_{FV}$ dimensional

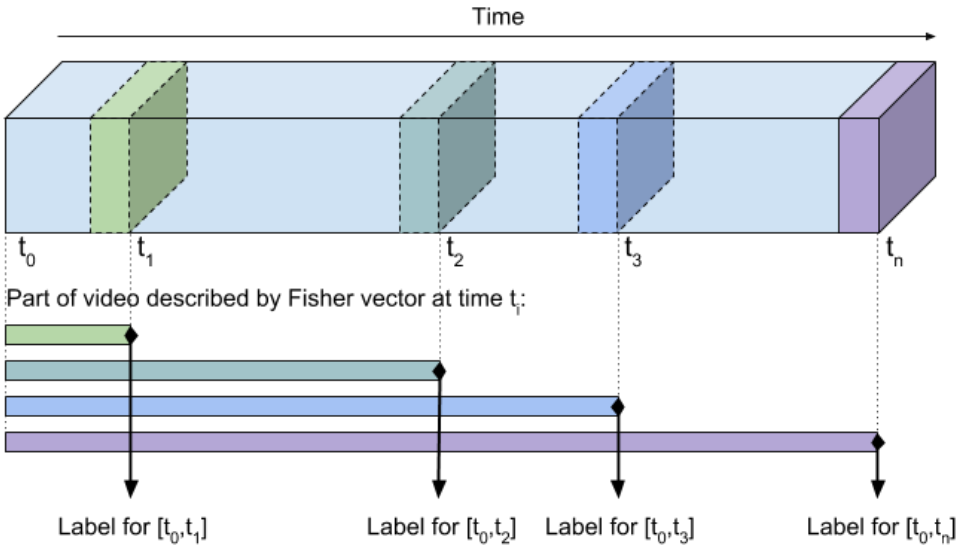


Figure 4.3: An illustration showing how our proposed model can be used to extract the Fisher vector representation of different parts of the video, allowing us to make predictions about the video label online, as new video frames arrive into the network. That is, at time t_1 we can predict a label for the interval $[t_0, t_1]$ and at time t_2 we can predict a label for the interval $[t_0, t_2]$. Also, if we reset the aggregated Fisher vector representation at time t_2, we could predict a label for the interval $[t_2, t_3]$ at time t_3.

matrix and a m-dimensional vector, where m is the number of classes and d_{FV} is the dimensionality of the FV layer output, $d_{FV} = K \cdot (2n_c + 1)$. In total, the top layers of our architecture contain $D \cdot (n_c + 1) + K \cdot (m \cdot (2n_c + 1) + 2n_c + 1) + m$ trainable parameters. As a concrete example, the top layers of the architecture we finetuned on the UCF-101 dataset (Table 4.3) with $n_c = 100$, $K = 256$, $D = 6144$ and $m = 101$ contained 5 869 157 trainable parameters.

The dimensionality of the last pooling layer in the VGG-16 [90] architecture for a single input is (512, 7, 7). The pooling layer is fully connected to 4096 units, followed by two more fully connected layers containing 4096 and 1000 units respectively. Including the biases, this corresponds to having 123 642 856 trainable parameters after the convolution and pooling layers. In case of the last fully connected layer having only 101 units (when applied to the UCF-101 dataset), the parameter count is

119 959 653. Similarly, the fully connected layers in the CNN-M-2048 [15, 89] network contain 4096, 2048 and 1000 units each, amounting to 85 941 224 trainable parameters in the top layers. For the case when there are 101 classes, the fully connected layers contain 84 099 173 trainable parameters. This is explained more in detail in Appendix C.2.

By replacing the fully connected layers at the end of the network with the layers we propose, the number of trainable parameters drops to under 5% of the original number in case of the VGG-16 network, and under 7% in case of using the CNN-M-2048 network. Details of the CNN-M-2048 architecture can be found in Appendix C.1.

4.1.9 Adding motion information

The methods described in this chapter so far only take into account features extracted from static frames and ignore the motion information which is highly important in the problem of action recognition. In order to improve the performance of our system we need to find a way of including some type of motion features. To this end, we will try to make use of optical flow (Figure 4.4) extracted from adjacent frames in the video, which was shown to improve the performance in a number of recent works on action recognition, including [89, 111, 71, 117].

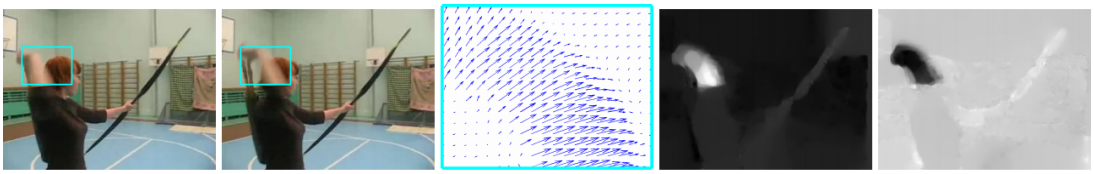


Figure 4.4: An illustration of optical flow borrowed from [89]. The first two images (a) and (b) show two adjacent frames from a video. (c) shows the optical flow in the area selected by the rectangle. (d) and (e) show the horizontal and vertical components of the optical flow respectively.

Optical flow between two frames is represented by a vector field of displacements, which can be seen as an image containing two channels, one containing the x compon-

ent of the displacement vectors, and the other containing the y component. In order to use both the static and dynamic features for action recognition, two different networks are usually combined, each dealing with one type of data. This kind of model was first introduced in [89].

4.2 Experiments and results

The UCF-101 dataset introduced in [93] consists of 13320 video clips from 101 different classes, divided into three pairs of train and test sets. To evaluate the performance of a method on this dataset, the average accuracy over the three splits is reported. We first run our experiments only on the first split and only evaluate the most promising approach on all three splits.

We start by implementing the method described in [76] in the Lasagne/Theano framework [23, 100] and treat it as the baseline for our experiments which we perform on the UCF-101 dataset. Training the architecture included training a single layer convolutional restricted Boltzmann machine [57] containing 64 filters of size 5×5px, learning a PCA projection ($n_c = 100$), training a GMM ($K = 256$) using the expectation-maximization algorithm and training a multi-class SVM classifier ($C = 100$). All these steps, except for training of the SVM are done in an unsupervised manner. After initializing the parameters of our architecture with the ones we got by the unsupervised training steps mentioned above, we did one epoch of finetuning of the whole network using the AdaGrad adaptive gradient algorithm [25].

The size of the temporal window, i.e. the number of frames needed to calculate a single FV descriptor, is set to $t = 15$ in all of our experiments. The size of the spatial window in the spatio-temporal pooling layer is set to correspond to a window of 32×32 pixels in the input video ($S_h = S_w = 7$, when the single convolutional RBM was used). These are the values used in other similar works, e.g. [106]. We can control how dense we want to sample the features from the given video by setting the spatial

stride parameter δ_S and the temporal stride parameter δ_T. In order to decrease the time needed to do a single finetuning pass through the training set, we use $\delta_S = 7$ "pixels" (corresponding to 16 pixels in the input video) and $\delta_T = 15$ frames in most of our finetuning experiments. One epoch of finetuning using these parameters on the whole training set takes around 10 hours on a Titan X GPU.

As can be seen from the results reported in Table 4.1, our method using features extracted from a single layer convolutional RBM performs better than the other state of the art methods shown in Table 4.5 that suffered from overfitting when trained only on the UCF-101 dataset. However, when more complex models are pretrained on datasets that provide larger amounts of data than the UCF-101 dataset, the performance of the simple single layer network is easily surpassed. This is not surprising as the simple network is too shallow to learn more discriminative features needed for action classification. To show how our proposed method works when the simple network is replaced with a more complex one, we choose the VGG-16 network from [90] pretrained on the ImageNet dataset, which was also used in the two-stream network of [27].

The VGG-16 network consists of 13 convolutional layers, followed by 3 fully connected layers. We first use the outputs of the conv4_3 layer as the input to the layers proposed in this work. Similar to the previously described experiment, we randomly extract 1000 subvolumes from conv4_3 layer's feature maps, corresponding to 32×32 px and $t = 15$ spatio-temporal subvolumes in the original video. A subset of 30 subvolumes per video is then used to learn a PCA mapping lowering their dimensionality to $n_c = 100$. These are then used to train a GMM with $K = 256$ components, which we use to extract Fisher vector descriptors from and finally train a SVM with $C = 100$. We report the results of this experiment on all three splits of UCF-101 in table 4.2. We repeat the same procedure replacing the conv4_3 layer by conv5_3 and report the results in Table 4.3. As the features extracted from the conv5_3 layer performed better

than the ones from layer conv4_3, we pick this layer for our finetuning experiments.

The larger network is more prone to overfitting so we regularize the finetuning using using dropout [94] ($p = 0.9$) on the output of the local feature extraction layers. To maximize the amount of information available in the network during both training and testing we set the spatial stride $\delta_S = 1$, but we keep the temporal stride fixed to $\delta_T = 15$ as in the previous experiments. We finetune the network using stochastic gradient descent with momentum (set to 0.9), showing the network one video at a time. The initial learning rate was set to 0.0001 and it was multiplied by a factor of 0.95 after each epoch. One epoch of finetuning took around 17 hours ($\sim$ 30 FPS). Testing ran at a speed of around 40 FPS.

We use the same pipeline to try and do action classification using features learned from optical flow. We extracted optical flow from all the UCF-101 videos using Epicflow [81]. As the VGG-16 network was pretrained on RGB images and optical flow only contains 2 channels, we extract features from optical flow data by setting the R-channel input to the network to the x component of the flow, the G-channel to the y component of the flow and the B-channel to zeros. The results of these experiments are shown in Table 4.4. When combining the Fisher vectors extracted from RGB frames with the ones extracted from optical flow, we do this by simply concatenating them. We show that normalizing the concatenated versions of the Fisher vectors leads to further boosts in classification performance.

Table 4.1: UCF-101 split 1 classification accuracy, using features from a single convolutional RBM trained only on UCF101.

Method	**Split 1**
Single CRBM + FV [76]	55.06%
Ours, single CRBM, random sampling	59.95%
Ours, single CRBM, dense sampling	60.37%
Our finetuned network (after 1 epoch)	61.67%

Table 4.2: UCF-101 classification accuracy, using the conv4_3 layer features from the VGG-16 network pretrained on ImageNet.

Method	**Split 1**	**Split 2**	**Split 3**	**Average**
Random sampling	70.84%	70.30%	70.81%	70.65%

Table 4.3: UCF-101 classification accuracy, using the conv5_3 layer features from the VGG-16 network pretrained on ImageNet. Finetuning was done using SGD with initial learning rate = 0.0001, momentum = 0.9 and extraction layer dropout p = 0.9.

Method	**Split 1**	**Split 2**	**Split 3**	**Average**
Random sampling	75.65%	76.06%	74.89%	75.54%
Dense sampling	75.55%	76.33%	74.35%	75.41%
Finetuned network (after 5 epochs)	76.39%	77.29%	77.30%	76.99%
Finetuned network (after 11 epochs)	79.12%	78.63%	76.35%	78.03%
Finetuned network (after 33 epochs)	81.84%	-	-	-

Table 4.4: UCF-101 classification accuracy, using the conv5_3 layer features extracted from optical flow using the VGG-16 network pretrained on ImageNet.

Method	**Split 1**
Random sampling OF	65.84%
Random sampling OF + random sampling RGB	82.97%
Random sampling OF + random sampling RGB + normalize	84.54%
Random sampling OF + dense sampling RGB	85.17%
Random sampling OF + dense sampling RGB + normalize	87.07%

Table 4.5: State of the art methods using only static features, trained and evaluated on UCF-101.

Method	**Split**	**Accuracy**	**Total parameters**
Slow fusion network [46]	all	41.3%	
Spatial CNN-M-2048 [89]	1	52.3%	$\sim$ 90.63 M
Single CRBM + FV [76]	1	55.06%	$\sim$ 5.33 M
Ours, single CRBM	1	**61.67**%	$\sim$ **5.33 M**

4.3 Discussion and conclusion

Compared to the work of [76], where the same kind of features and encoding were used and the extraction was performed on randomly selected subvolumes, our network is naturally capable of performing dense sampling, thus increasing the available information from the underlying video and improving the final classification performance. By performing dense sampling and finetuning the whole network an improvement to

Table 4.6: State of the art methods using only static features, pretrained on a larger dataset, finetuned and evaluated on UCF-101.

Method	**Pretrained on**	**Split**	**Accuracy**	**Top layers parameters**	**Total parameters**
Slow fusion network [46]	Sports 1M	all	65.4%	-	-
Encoding objects [41]	ImageNet	all	65.6%	-	-
Spatial CNN-M-2048 [89]	ImageNet	1	72.8%	84.1 M	90.63 M
Ours, VGG-16	ImageNet	1	**81.84**%	**5.87 M**	**20.58 M**
Spatial VGG-16 [27, 90]	ImageNet	1	82.61%	119.96 M	134.67 M

61.3% was achieved. This is explained by the fact that including more training data helps prevent overfitting.

Let us note that the lowest level of our network extracts features at frame level from intensity information alone, and therefore is not directly comparable to the full two stream network from [89], one stream of which is trained on optical flow that was extracted offline at the input. However, we do compare favorably with the spatial stream of the network when trained directly on static frames of the UCF-101 dataset, where it overfits and results in a classification accuracy of 52.3% - compared to 61.67% that we obtain when using a simple, single-layer convolutional RBM. In order to prevent overfitting, the two stream network is pretrained on a different, larger dataset - this alone improves the accuracy of [89] to 72.8%.

Our approach, which includes the time dimension by pooling feature maps extracted from subvolumes of the video, achieves an accuracy of 61.67%, without including optical flow explicitly and only using a single convolutional RBM at the lowest layers of the network. The work of [46] also tried to tackle the problem of including motion features implicitly by trying to learn them from stacks of static frames. The approach of slowly fusing the feature maps resulted in an accuracy of 65.4% on the UCF-101 dataset when pretrained on a larger (Sports 1M) dataset. Whereas training directly on UCF-101 resulted in overfitting with an accuracy of 41.3%. This is again comparable to the 61.67% that we obtain with the proposed approach, when using the simple

single-layer convolutional RBM features.

By simply replacing the single convolutional RBM layer at the lowest level of our architecture with VGG-16, a deep network containing 13 convolutional layers pretrained on ImageNet, we boost the classification accuracy to 75.41%. The main contribution of our proposed method is shown after performing the finetuning of the network as a whole, which further boosts the classification accuracy on UCF-101 to 81.84%. While this is lower than the 82.61% achieved by the VGG-16 network as the spatial stream of [27], we point out that our architecture contains 20.58 million trainable parameters in total, compared to the 134.67 million parameters contained in VGG-16. If we only look at the top layers of the two architectures, the 3 fully connected layers containing 119.96 million parameters in VGG-16 can be replaced by our proposed layers that contain only 5.87 million parameters, that is less than 5% of the parameter count, at the cost of diminishing the classification performance by 3.5%. Longer finetuning and a finer choice of the finetuning hyperparameters should lower this performance gap. On the other hand, our method compares favorably to the CNN-M-2048 spatial stream of [89], achieving 81.84% versus its 72.8%, while requiring less than 23% of its total trainable parameter count.

4.3.1 Conclusion

In this chapter we have proposed a convolutional architecture that expresses the various steps of the Fisher vector based action recognition as layers in convolutional neural network that can be trained or refined end to end in a supervised manner. Our model outperforms significantly the baseline architecture where the various levels are trained in a layer by layer manner unsupervised, and state of the art CNN architectures when trained on the same amount of data. We show that replacing the top fully connected layers in popular convolutional network architectures with our proposed layers results in a significant reduction of the needed trainable parameter count, while achieving a comparable performance, or even significantly surpassing the performance of similar

architectures.

Chapter 5

Semi-supervised Fisher vector network

Contents

In the previous chapter we have shown how the standard Fisher vector pipeline for classification consisting of a local feature extraction step, a PCA dimensionality reduction step, a step where a Gaussian mixture model is learned and the final step of training a classifier can all be viewed as a single neural network, allowing us the finetuning of the model in a supervised, end-to-end fashion. However, even when we had all the mentioned steps defined as network layers, we could start the finetuning only after the layers had been initialized with parameters learned using offline batch learning procedures that require access to the whole training set at once. That is, the initialization consisted of running PCA on the whole training set, followed by using expectation-maximization to fit to data of lower dimensionality with a GMM

and then training an SVM using Fisher vectors extracted using the GMM. After that the network was ready for finetuning.

What we are interested in exploring in this chapter are methods that would give us a way of training all the parameters in the Fisher vector pipeline in an online fashion, this way becoming suitable for applications where the training data is not available in advance, but is arriving in batches. Defining such a procedure would allow us to optimize our network using mini-batch gradient descent based methods, the usual methods used for training deep learning architectures. As the PCA dimensionality reduction step is optional in the Fisher vector pipeline, we will ignore it and first focus on defining a method that learns a Gaussian mixture model incrementally.

The final and the main goal of this chapter is to propose a method which allows incorporating unlabeled data into the process of the Fisher vector pipeline training, i.e. a method for learning a semi-supervised Fisher vector encoding of the input data. To test how the proposed method works on real data, we choose to apply it at an image classification problem; classifying tiny RGB images from the Cifar-10 [49] dataset into 10 classes, while artificially varying the amount of available labeled and unlabeled data. We also show how the same method can be used in an action recognition problem by running similar experiments on the UCF-101 dataset.

The main contributions of this chapter can be summarized as follows:

- We describe a method for fitting Gaussian mixture models that can be used when the training data is not available all at once, but it is arriving in mini-batches, i.e. the data is arriving in small subsets of the training set. The same method can be used for training of the GMM defined as a network layer, described in the previous chapter. Also, the method can be run on a GPU, leading to smaller training times compared to methods that can only run on CPUs.
- We define a method that leverages the availability of unlabeled data for training

of an improved, semi-supervised version of the Fisher vector encoding. Similarly as in the previous chapter, we define a network that extracts the Fisher vector descriptor of the input data, which we name the semi-supervised Fisher vector network. We perform a number of experiments which show how increasing the amount of unlabeled data helps improve the classification performance of our model at the problem of image classification (on CIFAR-10) and action recognition (on UCF-101).

We start by describing the problem of Gaussian mixture model fitting.

5.1 Gaussian mixture model training

We are faced with a problem of estimating a probability distribution of a random variable X for which we assume that it depends on another random variable Z, whose values we cannot observe, i.e. Z is a hidden random variable. If we parameterize the joint distribution of the two variables with a set of parameters $\boldsymbol{\theta}$, the distribution of X can then be written by marginalizing over Z as

$$p(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_{z} p(\boldsymbol{x}, z|\boldsymbol{\theta}). \tag{5.1}$$

Finding the maximum likelihood estimate parameters of the model above, that is finding the $\boldsymbol{\theta}$ which maximizes $L(\boldsymbol{\theta}) = \log p(\boldsymbol{x}|\boldsymbol{\theta})$, cannot be done as a direct maximum likelihood estimation by setting the derivatives $\frac{\delta}{\delta\theta} L(\boldsymbol{\theta})$ to zero, as this leads to entangled equations without a closed form solution. However, the parameters can be found in the case when the posterior distribution $p(z|\boldsymbol{x}, \boldsymbol{\theta})$ is known. This observation lead to the idea of the expectation-maximization (EM) algorithm [22] which offers a way of finding maximum likelihood parameters using an iterative approach.

Starting from some initial estimate of the parameters $\boldsymbol{\theta}_0$, the EM algorithm works by iteratively performing two steps at each time step t; the E-step which determines

the posterior distribution $\tilde{p}_t(z) = p(z|\boldsymbol{x}, \boldsymbol{\theta}_{t-1})$ using the parameter estimates from the previous time step $\boldsymbol{\theta}_{t-1}$, and the M-step which finds new estimates of the parameters $\boldsymbol{\theta}_t$ as the ones that maximize the expected value of the log likelihood $L(\boldsymbol{\theta})$ under the estimated posterior distribution from the E-step, $\tilde{p}_t$. That is, $\boldsymbol{\theta}_t$ is set to the $\boldsymbol{\theta}$ that maximizes $E_{\tilde{p}_t}[\log p(\boldsymbol{x}, z|\boldsymbol{\theta})]$. This procedure is guaranteed to never worsen the log likelihood of the data [22].

Following the work of [68], we can define a function $F(\tilde{p}, \boldsymbol{\theta})$ whose value is maximized or at least increased by both the E-step and the M-step of the EM algorithm as

$$F(\tilde{p}, \boldsymbol{\theta}) = E_{\tilde{p}}[\log p(\boldsymbol{x}, z|\boldsymbol{\theta})] + H(\tilde{p}), \tag{5.2}$$

with

$$H(\tilde{p}) = -E_{\tilde{p}}[\log \tilde{p}(\boldsymbol{x})]. \tag{5.3}$$

This function can also be written in terms of a Kullback-Liebler divergence between $\tilde{p}(z)$ and $p_{\boldsymbol{\theta}}(z) = p(z|\boldsymbol{x}, \boldsymbol{\theta})$ as:

$$F(\tilde{p}, \boldsymbol{\theta}) = -D(\tilde{p}||p_{\boldsymbol{\theta}}) + L(\boldsymbol{\theta}), \tag{5.4}$$

where

$$D(\tilde{p}||p_{\boldsymbol{\theta}}) = \sum_z \tilde{p}(z) \log \frac{\tilde{p}(z)}{p_{\boldsymbol{\theta}}(z)} \tag{5.5}$$

is the KL divergence and $L(\boldsymbol{\theta})$ is the log-likelihood, $L(\boldsymbol{\theta}) = \log p(\boldsymbol{x}|\boldsymbol{\theta})$. It was shown in [68] that the local and global maxima of $F(\tilde{p}, \boldsymbol{\theta})$ are at the same time the local and global maxima of $L(\boldsymbol{\theta})$, so maximizing $F(\tilde{p}, \boldsymbol{\theta})$ also leads to the maximum likelihood estimate parameters $\boldsymbol{\theta}$ which we want to find. When the objective function is defined as in Equation 5.2 the E-step of the EM algorithm maximizes $F(\tilde{p}, \boldsymbol{\theta})$ with respect to $\tilde{p}$ and the M-step maximizes it with respect to $\boldsymbol{\theta}$.

5.2 Online Gaussian mixture model training

In the case when the expression $E_{\tilde{p}_t}[\log p(\boldsymbol{x}, z|\boldsymbol{\theta})]$ which we wish to optimize in the maximization step of the EM algorithm is not completely maximized, but only moved

towards its maximum, the procedure will still result in improvements of the data likelihood [22]. Algorithms that perform the maximization step only partially are called generalized expectation-maximization (GEM) algorithms and are the type of algorithms that we are interested in for our problem.

As our goal is to have a way of training a GMM incorporated in a neural network as a layer, we want to be able to use the usual, gradient based optimization methods for learning the GMM's parameters. The approach which we will take for implementing the M-step will be to simply calculate the gradients of the function defined in Equation 5.4 with respect to all of the GMM parameters and perform a number of gradient ascent steps to move the value of $F(\tilde{p}, \boldsymbol{\theta})$ toward its maximum.

Here we repeat the definition of the Gaussian mixture model, previously described in Subsection 4.1.4 as a sum of K weighted Gaussians:

$$u_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{k=1}^{K} w_k u_k(\boldsymbol{x}), \tag{5.6}$$

with

$$u_k(\boldsymbol{x}) = \frac{1}{(2\pi)^{\frac{D}{2}} |\boldsymbol{\Sigma}_k|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_k)' \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_k)\right) \tag{5.7}$$

being the probability density function of a single Gaussian distribution. In order to meet the constraints that the weights w_k need to be positive and sum up to one, we rewrite w_k in terms of internal weights α_k as $w_k = \frac{\exp(\alpha_k)}{\sum_l^K \exp(\alpha_l)}$. The posteriors or responsibilities describing how likely it is that a sample $\boldsymbol{x}$ was generated by the k-th mixture component can be calculated as:

$$\gamma(k) \equiv p(Z = z_k|\boldsymbol{x}) = \frac{w_k u_k(\boldsymbol{x})}{\sum_l^K w_l u_l(\boldsymbol{x})}. \tag{5.8}$$

We represent the set of the GMM parameters as $\boldsymbol{\theta} = \{\alpha_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, k = 1, ..., K\}$, where α_k is the k-th internal component weight, $\boldsymbol{\mu}_k$ is its mean vector and $\boldsymbol{\Sigma}_k$ its covariance matrix. We will denote the set of GMM parameters at iteration t as $\boldsymbol{\theta}_t$.

Finally, we will describe the steps of the generalized mini-batch EM algorithm which we use in this chapter. Given a mini-batch of training samples, i.e. only a small subset

of the available training data, with t denoting the iteration of the algorithm, we can summarize the algorithm as follows:

- E-step: Maximize $F(\tilde{p}_t, \boldsymbol{\theta}_{t-1})$ with respect to $\tilde{p}_t$ by setting $\tilde{p}_t(z_k) \leftarrow \gamma_{t-1}(k)$, where $\gamma_{t-1}(k)$ is the posterior distribution over all the samples in the current mini-batch, calculated using the parameters $\boldsymbol{\theta}_{t-1}$ from the previous time step, $t-1$.
- M-step: Repeat n_M times:
  - Calculate the gradients of $F(\tilde{p}_t, \boldsymbol{\theta}_{t-1})$ with respect to each of the parameters θ_{t-1} in $\boldsymbol{\theta}_{t-1}$; $\nabla_{\theta_{t-1}} F(\tilde{p}_t, \boldsymbol{\theta}_{t-1})$.
  - Update each θ_t in $\boldsymbol{\theta}_t$ using a gradient based method, e.g. $\theta_t \leftarrow \theta_{t-1} + \lambda \nabla_{\theta_{t-1}} F(\tilde{p}_t, \boldsymbol{\theta}_{t-1})$.

Note that the gradient based optimization method used in the M step is not limited to gradient ascent, but other methods such as Adagrad [25] or RMSProp [101] could be used too.

5.2.1 Experiments on a 2D toy dataset

In order to evaluate the performance of the method proposed above for online training of Gaussian mixture models, we first use an implementation of the standard batch expectation-maximization algorithm [22] from the scikit-learn [77] library and apply it to a toy problem in which we want to model a distribution of a set of points in two-dimensional space.

To generate the toy dataset we first randomly pick K means, $\boldsymbol{\mu}_k$, and K standard deviations, $\boldsymbol{\sigma}_k$, and use them to parameterize a set of K normal distributions $\{\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k^2), k \in [0, K-1]\}$. We then define a multinomial distribution parameterized by $\{\pi_0, \ldots, \pi_{K-1}\}$, with $\sum_{k=0}^{K-1} \pi_k = 1$, where each π_k represents the probability

of picking the k-th normal distribution for sampling. We finally generate a set of n two-dimensional vectors by choosing one of the K normal distributions according to the multinomial distribution and sampling from it. This results in data grouped in K random clusters whose means and variances are known. By using the described procedure we can generate arbitrary many random point distributions, allowing us to compare the performance of our method to scikit-learn's implementation of the batch EM algorithm. Examples of the generated toy datasets are shown in Figures 5.1a and 5.2a.

Parameter initialization

To be able to directly compare the two methods we run both of them from the same parameter initialization. We use the k-means++ algorithm [5], a method used for initializing the locations of centroids for the k-means algorithm, to initialize the means $\boldsymbol{\mu}_k$ for each of the Gaussians in our mixture model. We initialize each of the K component weights w_k to $1/K$ and set the standard deviations $\boldsymbol{\sigma}_k$ to fixed values.

Results

We generated a number of random 2D toy datasets consisting of 10 000 training and 10 000 testing samples from 32 different Gaussian distributions. We show the generated datasets in the top left corners of Figures 5.1, 5.2 and 5.3. We then use the implementation of the batch EM algorithm from the scikit-learn library to fit a GMM with 32 components, starting from the intialization described above. The training was stopped after the improvement of the training data log likelihood between two iterations fell below a certain threshold value.

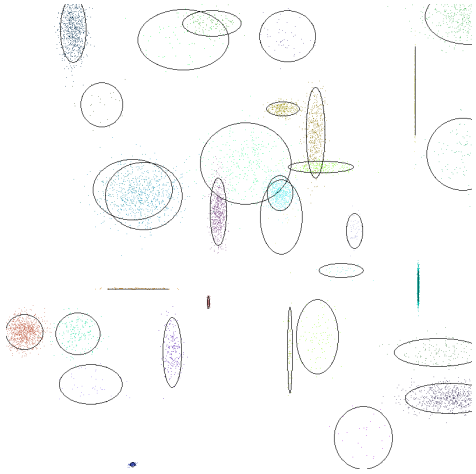
The results acquired using the batch EM algorithm are shown in top right corners of Figures 5.1, 5.2 and 5.3. After that we run the version of the EM algorithm we described in Section 5.2 using different gradient based optimization methods, different learning rates and different number of M-step iterations. We run the training for a

fixed number of iterations. In the first two experiments the number of samples given in a mini-batch was the same as the total number of available training samples. The results of running the proposed method are shown in the bottom row of Figures 5.1, 5.2. In the last experiment we use a mini-batch size of 500 and show the results in the bottom row of Figure 5.3. We also report the log likelihood of the testing data in each of the experiments below the figures. It can be seen that the learned Gaussians identified most of the groundtruth clusters. The proposed algorithm also learned some Gaussians with a small variance, which also increased the average likelihood. The values of the variances during training were limited to not fall bellow a predefined value.

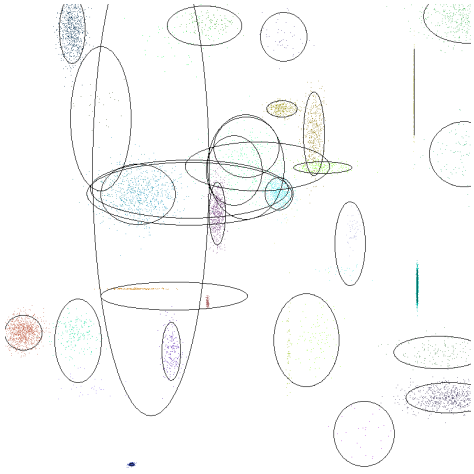
5.3 Semi-supervised Fisher vector encoding

In this section we will define a method for leveraging unlabeled data in the Fisher vector classification pipeline. An illustration of the architecture that we describe in this chapter is shown in Figure 5.4.

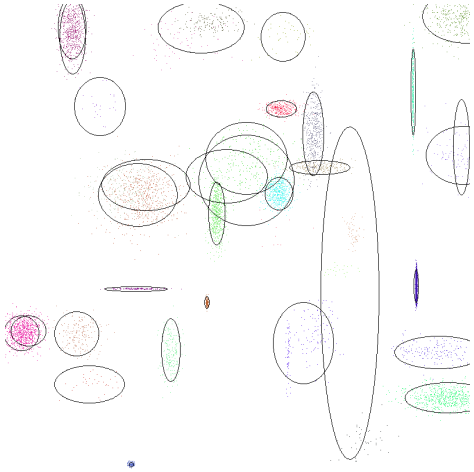
Having a method for training Gaussian mixture models with a gradient based optimization method which we described in the previous section (Section 5.2) gives us a natural way to build a hybrid network that is trained with the goal of optimizing two different objective functions; an unsupervised objective function and a supervised objective function. The unsupervised objective function is the one whose goal is to fit a Gaussian mixture model to the given data and the supervised objective function would be the one that forces the network to output the correct label given a training sample. We already have both of the needed functions defined - the unsupervised objective function, C_{unsup} is $F(\tilde{p}, \boldsymbol{\theta})$ from Equation 5.2 and the supervised objective function, which we will denote as C_{sup}, is the squared hinge loss in Equation 4.15,



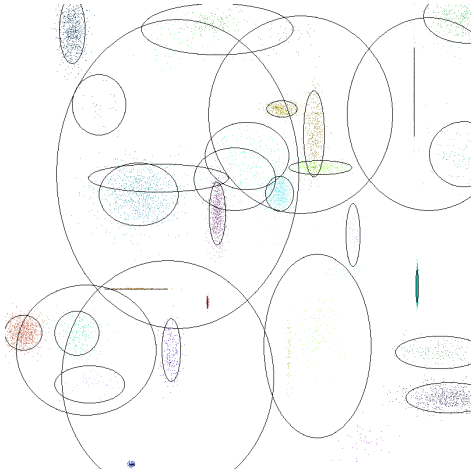
(a) Groundtruth data generated by sampling from 32 Gaussians.



(b) GMM learned using scikit-learn batch EM implementation. Mean $\log p(x_{test}) = -13.48343$.

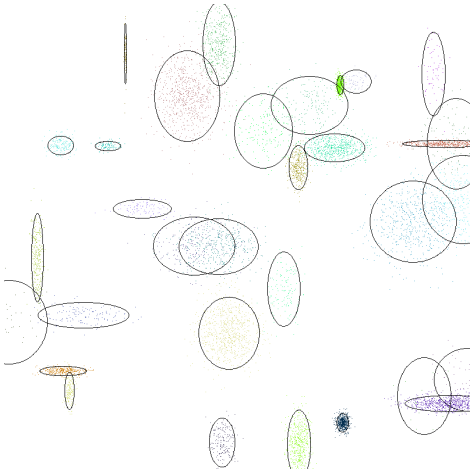


(c) GMM learned using the proposed algorithm with Adagrad, 1 000 M-step iterations, learning rate 1. Mean $\log p(x_{test}) = -10.9322$.

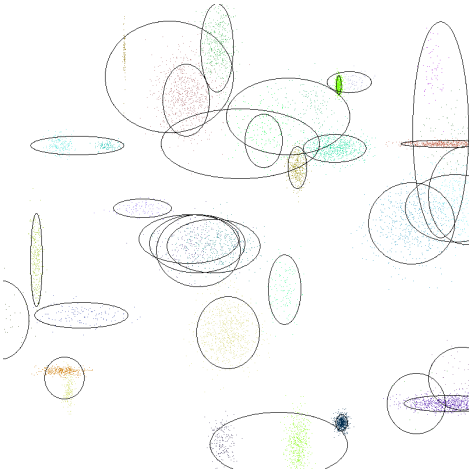


(d) GMM learned using the proposed algorithm with Nesterov momentum, 1 000 M-step iterations, learning rate 10. Mean $\log p(x_{test}) = -10.9354$.

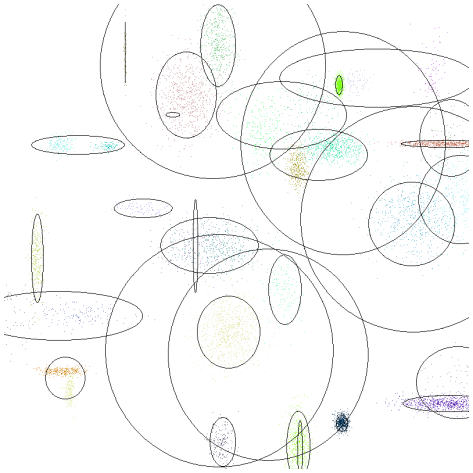
Figure 5.1: Experiments on the 2D toy dataset. The first subfigure shows the generated dataset, the second subfigure shows the GMM learned using scikit-learn's EM implementation, the third subfigure shows the GMM learned using the proposed EM algorithm with Adagrad and the fourth subfigure shows the GMM learned using the proposed EM algorithm with Nesterov mementum. The training set contained 10 000 samples and the mini-batch size in the last two experiments was set to 10 000.



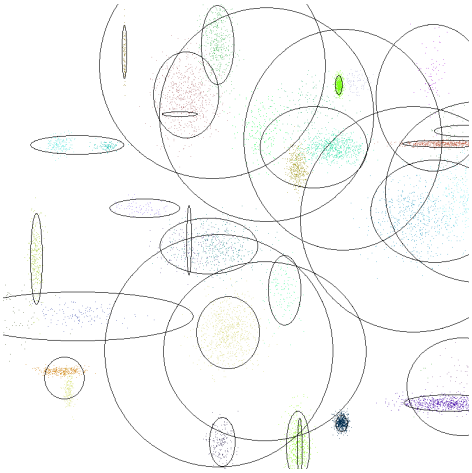
(a) Groundtruth data generated by sampling from 32 Gaussians.



(b) GMM learned using scikit-learn batch EM implementation. Mean $\log p(x_{test}) = -14.46651$.

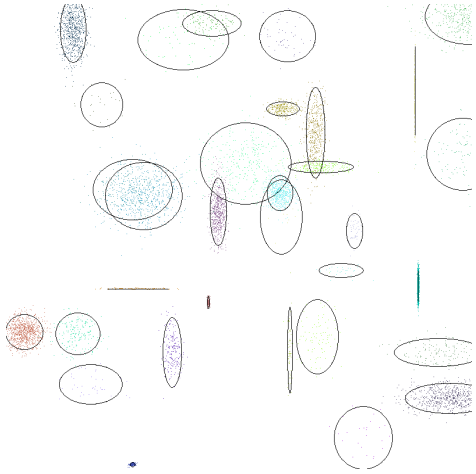


(c) GMM learned using the proposed algorithm with Nesterov momentum, 1 000 M-step iterations, learning rate 10. Mean $\log p(x_{test}) = -11.76504$.

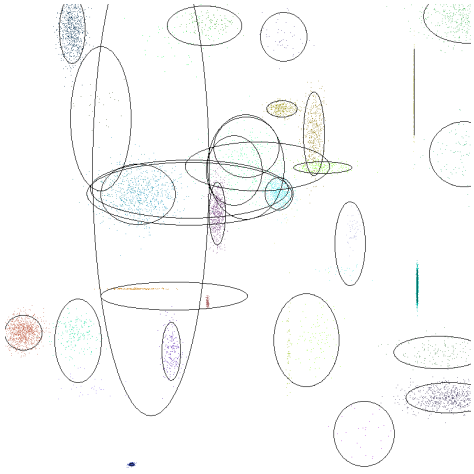


(d) GMM learned using the proposed algorithm with RMSProp, 1 000 M-step iterations, learning rate 0.01. Mean $\log p(x_{test}) = -11.80651$.

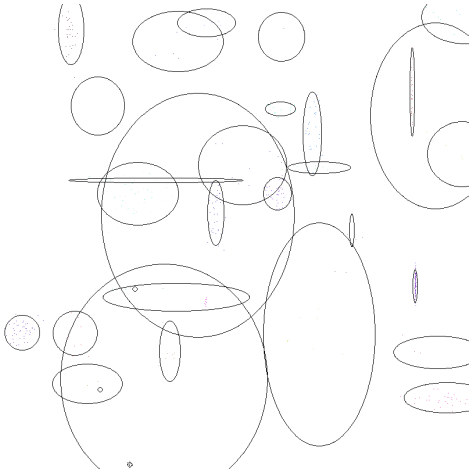
Figure 5.2: Experiments on the 2D toy dataset. The first subfigure shows the generated dataset, the second subfigure shows the GMM learned using scikit-learn's EM implementation, the third subfigure shows the GMM learned using the proposed EM algorithm with Nesterov momentum and the fourth subfigure shows the GMM learned using the proposed EM algorithm with RMSProp. The training set contained 10 000 samples and the mini-batch size in the last two experiments was set to 10 000.



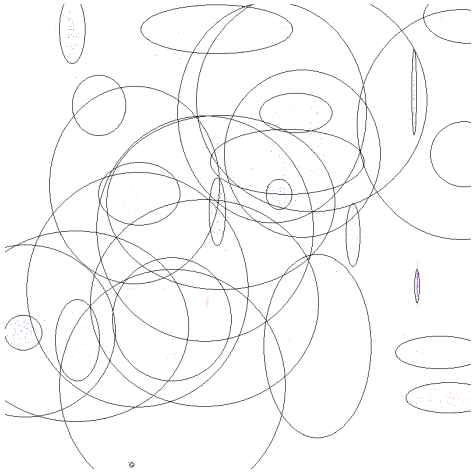
(a) Groundtruth data generated by sampling from 32 Gaussians.



(b) GMM learned using scikit-learn batch EM implementation. Mean $\log p(x_{test}) = -13.48343$.



(c) GMM learned using the proposed algorithm with Adagrad, 100 M-step iterations, learning rate 1. $\log p(x_{test}) = -11.03016$.



(d) GMM learned using the proposed algorithm with Adagrad, 10 M-step iterations, learning rate 1. $\log p(x_{test}) = -11.14458$.

Figure 5.3: Experiments on the 2D toy dataset. The first subfigure shows the generated dataset, the second subfigure shows the GMM learned using scikit-learn's EM implementation, the third and the fourth subfigures show the GMM learned using the proposed EM algorithm with Adagrad, but with different numbers of M-step iterations. The training set contained 10 000 samples and the mini-batch size in the last two experiments was set to 500.

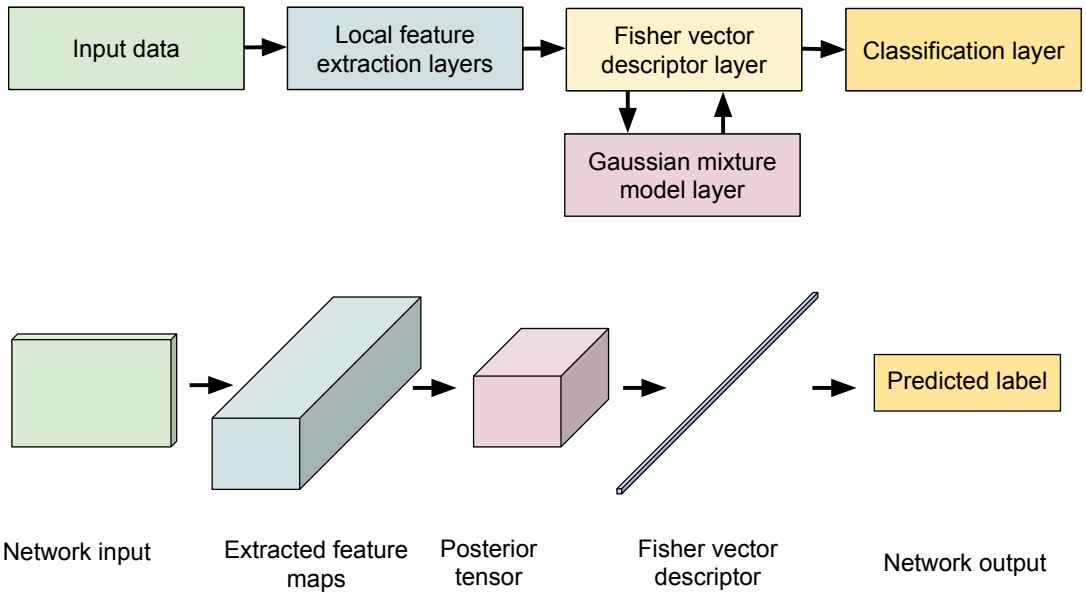


Figure 5.4: An illustration of the proposed architecture. It is a simplified version of the network proposed in Chapter 4, without the spatio-temporal pooling layer and the dimensionality reduction layer. For example, we can think of the network input as an RGB image that is fed through local feature extraction layers that give some feature maps as output. The "pixels" of the feature maps are modeled with a GMM layer that outputs a tensor of posteriors for each "pixel" in the given feature map. Feeding the extracted feature maps and the posterior tensor into the Fisher vector descriptor layer gives the FV descriptor as an output. The FV descriptor is then fed into the classification layer that outputs a class prediction for the image given at the input of the network.

from Section 4.1.6:

$$C_{unsup} \equiv F(\boldsymbol{x}, \tilde{p}, \boldsymbol{\theta}) = -\sum_{z} \tilde{p}(z) \log \frac{\tilde{p}(z)}{p_{\boldsymbol{\theta}}(z)} + \log p(\boldsymbol{x}|\boldsymbol{\theta}), \tag{5.9}$$

and

$$C_{sup} \equiv C(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w}, \boldsymbol{b}) = \frac{\lambda}{2}||\boldsymbol{w}||^2 + \sum_{j}^{m} \max\left(0, 1 - y_j \cdot s_j\right)^2, \tag{5.10}$$

where $\boldsymbol{s} = \boldsymbol{x}\boldsymbol{w}^T + \boldsymbol{b}$ is the SVM score, s_j denotes the j-th element of $\boldsymbol{s}$ and $\boldsymbol{y}$ is the label l of sample $\boldsymbol{x}$ encoded as a vector where, in case of dealing with m classes, $m-1$ elements are set to -1 and a single element at position l is set to 1. After this, we can

define the combined objective cost as

$$C_{hybrid} = C_{sup} - \lambda \cdot C_{unsup}, \tag{5.11}$$

where λ is a weighting coefficient. We subtract the unsupervised cost in order to turn its optimization from a maximization into a minimization problem.

Trying to optimize the local feature extraction layers' weights jointly with the GMM and SVM layers is not straightforward as the feature extraction layers' weights might collapse to zeros, leading to a trivial but not meaningful solution that minimizes the unsupervised part of the cost. Here we focus only at the GMM and SVM layers and keep the parameters of the local feature extraction layers fixed. The training of the proposed model can in this case be done as follows.

We loop through the training set consisting of both labeled and unlabeled data, at each step taking a mini-batch from the set. Each chosen mini-batch can either consist of a mix of labeled and unlabeled samples, all the samples can be unlabeled or all the samples can be labeled. For each mini-batch we first perform the E-step of the EM algorithm described in Section 5.2, setting $\tilde{p}$ to the one that minimizes $-\lambda \cdot C_{unsup}$ while keeping the other parameters fixed. This is followed by n_M M-steps in which all the GMM parameters θ from $\boldsymbol{\theta}$ are updated using a gradient-based optimization method. Finally, we update all the parameters of both the GMM and the SVM layers by taking gradients of the combined cost, C_{hybrid}, with respect to each of the parameters and moving towards its minimum again using a gradient-based optimization method. In the case when a sample is unlabeled, its supervised part of the cost is set to 0. We evaluate the proposed method in the following section.

5.4 Experiments and results

In this section we report how we evaluated the proposed semi-supervised Fisher vector network on the problems of image classification and action recognition.

5.4.1 Image classification on CIFAR-10

The CIFAR-10 dataset [49] is a standard image classification dataset containing small 32×32 px images from 10 different categories. Some examples of the images from the dataset can be seen in Appendix B.2. In our experiments the images are preprocessed by subtracting the mean calculated on the training set from each image.

The architecture that we will use for the experiments we run on the CIFAR-10 dataset is shown in Figure 5.4, where the local feature extraction layers were replaced by the layers taken from the VGG-16 network [90], pretrained on the ImageNet dataset. The structure of the VGG-16 network can be found in Appendix C.2. In each experiment we will specify which layer is used as the input layer to the GMM layer.

Architecture without a GMM layer

The first experiments that we run are with an architecture that does not contain a GMM layer, that is, the outputs of the VGG-16 network layer are fed straight into the SVM layer. First we keep the VGG layer weights fixed and only train the SVM layer for a fixed number of epochs. We then try training the whole network using the SVM cost (Equation 5.10). We check how adding an additional convolutional layer with 512 1×1 px filters affects the accuracy. We also try adding an additional fully connected layer with 16416 units. We use different layers of VGG-16 as the input to the SVM layer. The results are shown in Table 5.1.

Table 5.1: Classification results on the CIFAR-10 test set using different outputs of the VGG-16 network directly fed into the SVM layer. We train the network using RMSProp for 100 epochs.

Output layer	**Output shape**	**Parameters trained**	**Accuracy**
conv_3_3	(128, 256, 8, 8)	SVM	69.96%
conv_3_3	(128, 256, 8, 8)	all	81.97%
conv_3_3 + conv layer	(128, 32768)	all	81.99%
conv_3_3 + FC layer	(128, 16416)	all	82.33%
conv_4_3	(128, 512, 4, 4)	all	85.44%
conv_5_3	(128, 512, 2, 2)	all	86.39%

Architecture with a GMM layer trained offline

In these experiments we pass images from the training set through the VGG network and model the outputs of its different layers with a GMM trained using an implementation of the EM algorithm from the scikit-learn library. Once the GMM parameters have been learned, we extract Fisher vector representations of the input images and use them to train a classifier by optimizing the SVM squared hinge loss. We use mini-batch gradient descent with momentum to do the training for 100 epochs, with the learning rate set to 0.001, momentum 0.9 and mini-batch size of 128. The number of GMM components was 64.

Table 5.2: Classification results on the CIFAR-10 test set using different outputs of the VGG-16 network. The GMM layer contained 64 components and was trained using scikit-learn's implementation of the batch EM algorithm. The SVM was trained using gradient descent with momentum for 100 epochs.

Output layer	**Output shape**	**FV size**	**Parameters trained**	**Accuracy**
pool_5	(128, 512, 1, 1)	(128, 65600)	SVM	53.79%
conv_5_3	(128, 512, 2, 2)	(128, 65600)	SVM	57.10%
conv_4_3	(128, 512, 4, 4)	(128, 65600)	SVM	72.48%
conv_3_3	(128, 256, 8, 8)	(128, 32832)	SVM	77.73%
conv_3_3	(128, 256, 8, 8)	(128, 32832)	all	86.03%

Architecture with a GMM layer trained using the proposed hybrid objective function

In these experiments we will evaluate the proposed method for training a semi-supervised Fisher vector network, as described in Section 5.3. We are given a training set consisting of a fixed number of labeled samples and we vary the number of unlabeled data throughout the experiments. We start by initializing the GMM layer using the k-means++ algorithm [5], as it was done in Section 5.2.1. Only the labeled data is used to do the initialization.

The optimization method we use for the M-step is Adagrad with the initial learning rate set to 1. The updates with respect to the supervised objective function are

done using RMSProp using an initial learning rate that we mention for each of the experiments.

In the first experiment on the CIFAR-10 dataset we fix the number of labeled samples to 10 000 and change the amount of unlabeled samples from 0, 1 000, 5 000, 10 000, 20 000 to 40 000. The learning rate is set to 0.0001 and we perform 5 M-step iterations for each mini-batch. The training charts for this experiment can be seen in Figure 5.5.

In the second experiment on the CIFAR-10 dataset we fix the number of labeled samples to 1 000 and change the amount of unlabeled samples from 0, 1 000, 5 000, 10 000, 20 000 to 49 000. The learning rate is set to 0.0001 and we perform only a single M-step iteration for each mini-batch. The training charts for this experiment can be seen in Figure 5.6. In all of the mentioned experiments the coefficient λ was set to 0.1.

5.4.2 Action recognition on UCF-101

In order to evaluate how the semi-supervised Fisher vector network performs on an action recognition problem when the amount of unlabeled data is being increased we do the following experiments. We use the representation illustrated in Figure 3.3 in Chapter 3, using the layer conv5_3 from the VGG-16 network as the feature extraction layer. We extract 1 000 subvolumes from each video and pool them spatially and temporally as illustrated in the mentioned figure. As we are not interested in learning the PCA dimensionality reduction mapping, we use the PCA learned in the experiments from Chapter 4 to lower the dimensionality of each extracted subvolume to 100. We then end up with 1 000 feature vectors of size 100 for each video, which is what we feed into our GMM layer. The rest of the training process is the same as with the experiments we performed on the CIFAR-10 dataset. We also start by initializing the GMM using the k-means++ algorithm on the labeled part of the training data. The optimization method used in the M-step is Adagrad with the initial learning rate

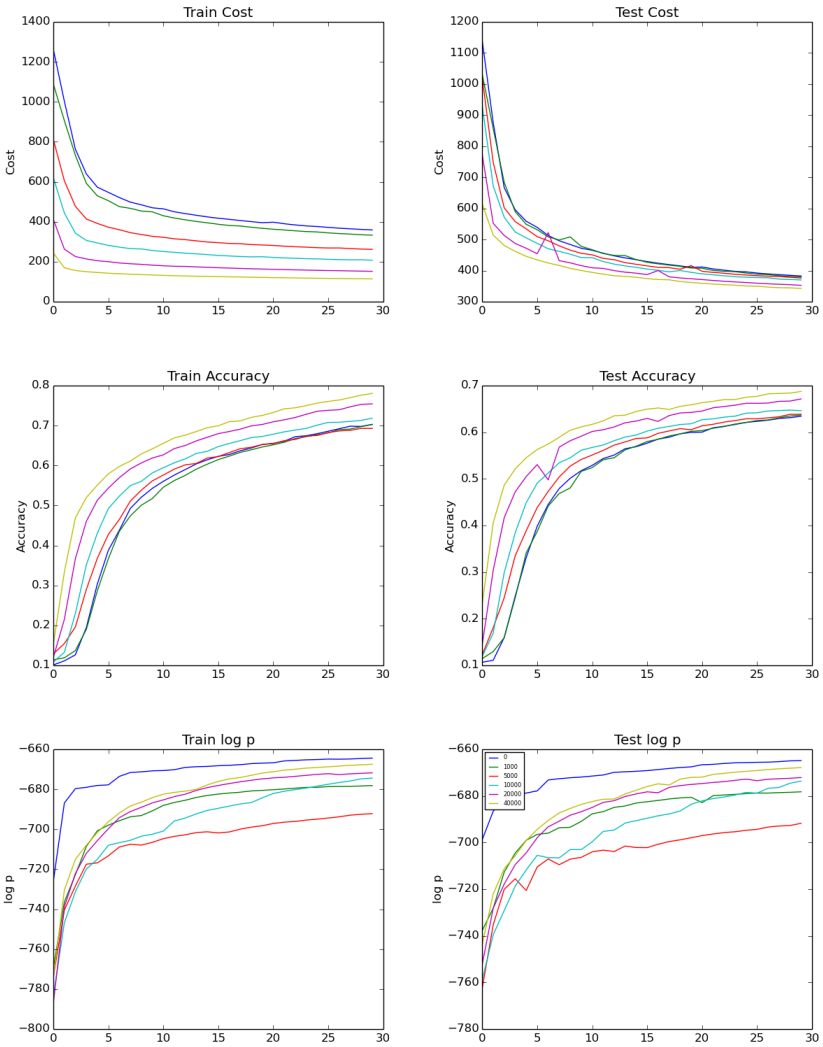


Figure 5.5: Training of the semi-supervised Fisher vector network on CIFAR-10 using 10 000 labeled samples and changing the number of unlabeled samples. The training was run for 30 epochs, the learning rate was 0.0001 and we used 5 M-step iterations.

set to 1, and the optimization method used to update the parameters with respect to the supervised objective function is RMSProp, as it was in the CIFAR-10 experiments.

We start by using 1 000 labeled samples and a varying amount of unlabeled samples (0, 1 000, 5 000 and 8 500) to train the semi-supervised Fisher vector network with 64, 128 and 256 GMM components using the proposed method for 100 epochs. We show the effect of using different amount of unlabeled data for the network with 64 GMM components in 5.7. Very similar behavior was observed when 128 or 256 GMM components were used, so we do not include the corresponding figures. In all of the mentioned experiments the coefficient λ was set to 0.1. We show the achieved classification performance from these experiments in Table 5.3.

Table 5.3: Classification results on the UCF-101 test set using semi-supervised Fisher vector networks with different numbers of GMM components. The number of labeled samples was fixed to 1 000, $\lambda = 0.1$ and the training ran for 100 epochs.

Number unlabeled samples	**64 GMM components**	**128 GMM components**	**256 GMM components**
0	42.21%	41.76%	42.62%
1 000	42.65%	43.73%	43.51%
2 000	45.65%	43.78%	44.59%
5 000	45.76%	45.76%	47.03%
8 500	46.41%	46.16%	46.32%

In the next experiment we chose to train the GMM component of the network offline, using the standard batch EM algorithm using both unlabeled and labeled data, and then we finetune the network using the labeled data only. We do this in order to be able to compare the achieved performance to the performance of the proposed semi-supervised training that updates all the network parameters at the same time, in an online fashion. The results shown in Table 5.4 demonstrate that the performance of the network where the GMM part was trained offline almost always surpasses the performance of the network trained in an online fashion. This is an expected result as the batch EM algorithm has access to the information from the whole training set at each step of the training. On the other hand, our online version can only make updates to the network parameters based only on the information contained in the current mini-batch.

Table 5.4: Classification results on the UCF-101 test set using semi-supervised Fisher vector networks with different numbers of GMM components. The GMMs are trained offline using labeled and unlabeled data and the networks are finetuned using the 1 000 labeled samples. The training ran for 100 epochs.

Number unlabeled samples	64 GMM components	128 GMM components	256 GMM components
0	46.70%	46.08%	43.76%
1 000	46.78%	46.05%	44.16%
2 000	46.08%	45.67%	44.11%
5 000	47.41%	46.83%	44.64%
8 500	45.94%	46.19%	44.70%

In order to see how much information is gained by increasing the available number of labeled samples compared to when the number of unlabeled samples is increased, we run a new set of experiments in which we do not use any unlabeled data, we only modify the amount of available labeled data. The results of this experiment are shown in Table 5.5. When comparing to the results of the previous experiment shown in Table 5.3 we can see that we only needed to add on average 600 labeled samples to surpass the performance of the semi-supervised Fisher vector network that was trained with 1 000 labeled and 8 500 unlabeled samples.

Table 5.5: Classification results on the UCF-101 test set using the semi-supervised Fisher vector network trained without any unlabeled data and different amounts of labeled data.

Number labeled samples	Test accuracy
1 000	42.21%
1 200	42.47%
1 400	44.67%
1 600	47.12%

In order to see how the performance of the proposed network changes as a function of the mini-batch size used during training, we run a set of experiments where we trained the network in a semi-supervised way while changing the size of the mini-batch. We trained the network using mini-batch size 1, 5, 50, 100, 250 and 500. We show the results in Table 5.6. It can be seen that, out of the different mini-batch sizes we have

used for the experiment, the best performance is achieved using a mini-batch size of 50.

Table 5.6: Classification results on the UCF-101 test set using the semi-supervised Fisher vector network trained using different mini-batch sizes. The numbers of both labeled and unlabeled samples were fixed to 1 000, we used 64 GMM components and the learning rate was set to 0.001.

Mini-batch size	**Test accuracy**
1	12.24%
5	37.39%
50	46.31%
100	42.65%
250	39.44%
500	32.29%

5.5 Discussion and conclusion

In the first experiments of this chapter (Table 5.1) we have shown how training an SVM using different output layers of the VGG-16 network affects the classification accuracy evaluated on the CIFAR-10 dataset. We can notice that using higher layers improved the performance as these layers capture more high-level features making it easier for the SVM to discriminate between the classes.

We have also shown that finetuning the whole network leads to significant improvements in the performance. This is expected as the VGG-16 network that we use was pretrained on a different dataset (ImageNet), so the weights are not tailored for the CIFAR-10 dataset.

In the experiments where a Fisher vector encoding was used (Table 5.2) we showed that having feature maps of larger spatial size results in better performance. This is because the Fisher vector combines local features into a global descriptor and having feature maps with an image structure provides more local information than when the whole image is represent as only a single vector as it is the case when e.g. the pool_5 layer is used.

As can be observed in Figure 5.5, where the proposed hybrid objective function was used to train the network for the image classification problem on CIFAR-10, increasing the amount of unlabeled samples resulted in increased classification performance. The highest test accuracy was achieved when 40 000 unlabeled samples were used (yellow line). The same can be seen in Figure 5.7 and Table 5.3, where different amounts of unlabeled data were used to train the network for the action recognition problem on UCF-101. The highest test accuracy in this case was also achieved when the largest number of unlabeled samples was used during training. The effect of using a small number of M-step iterations in our algorithm are shown in Figure 5.6 where only a single iteration is used. It can be seen that the improvement in the training is not as stable as when more iterations are used as shown in e.g. Figure 5.5 and Figure 5.7 (both using 5-M step iterations).

In Table 5.4 we show that the performance of the network when the GMM part of it is trained offline surpasses the performance of the network trained using our semi-supervised online approach. As already mentioned, this is because the batch EM-algorithm used to train the GMM offline has access to all the data during each EM iteration, while our online algorithm only sees a single mini-batch at a time. We note that the batch EM cannot be used in all situations - all data might not be available immediately, or it might be impossible to apply the batch algorithm because of memory constraints. Our proposed training method does not have this problem.

We tried matching the performance of the network trained in a semi-supervised way with a network trained only with labeled data in an experiment shown in Table 5.5. We show that only 600 additional labeled samples were needed to match the performance gained from adding 8 500 unlabeled samples.

We checked how the performance changes as a function of training time mini-batch size, shown in Table 5.6. It can be seen that the smallest mini-batch size resulted in the worst performance, as a single sample does not bring much information about

the distribution being modeled. If the mini-batch is too large the performance again degrades, as it is often observed when training neural networks.

To conclude, we presented a novel semi-supervised Fisher vector network and showed can be applied on an image classification problem (on CIFAR-10) and an action recognition problem (on UCF-101). We believe that we have shown some promising results and further experiments should be performed to see how well the semi-supervised Fisher vector network would perform on the whole UCF-101 dataset when unlabeled data is added from a larger dataset, e.g. the YouTube-8M dataset [1].

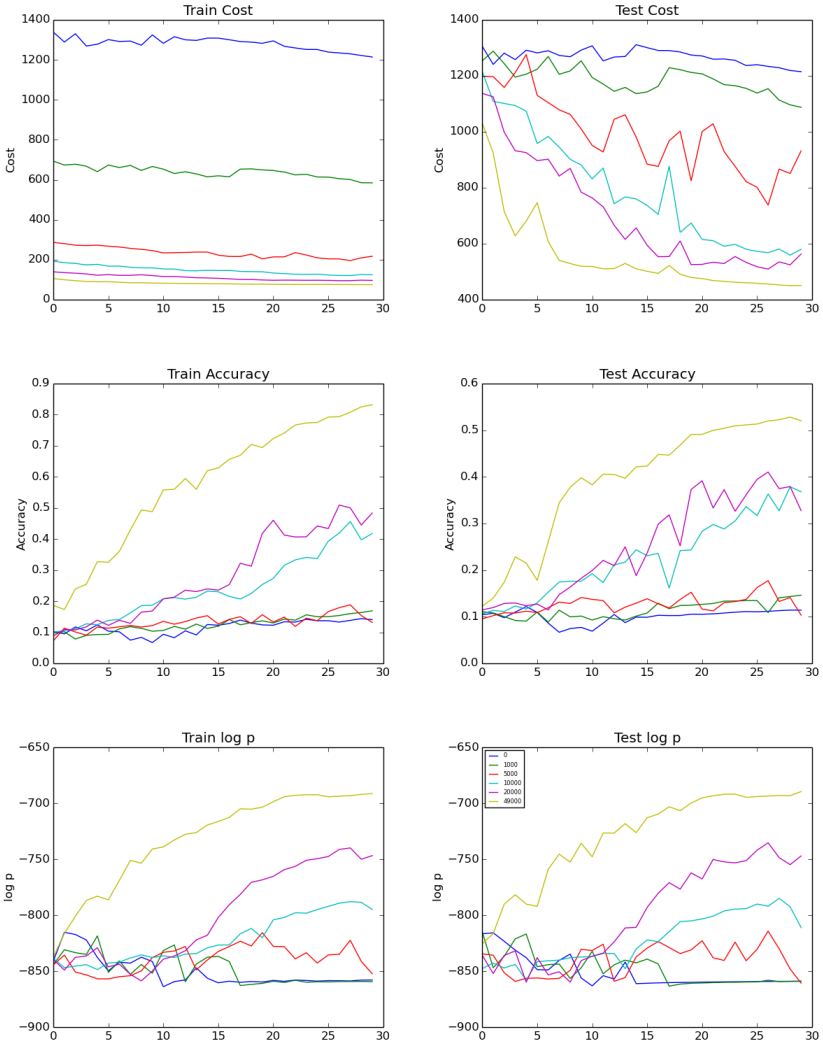


Figure 5.6: Training of the semi-supervised Fisher vector network on CIFAR-10 using 1 000 labeled samples and changing the number of unlabeled samples. The training was run for 30 epochs, the learning rate 0.0001 and we used only a single M-step iteration.

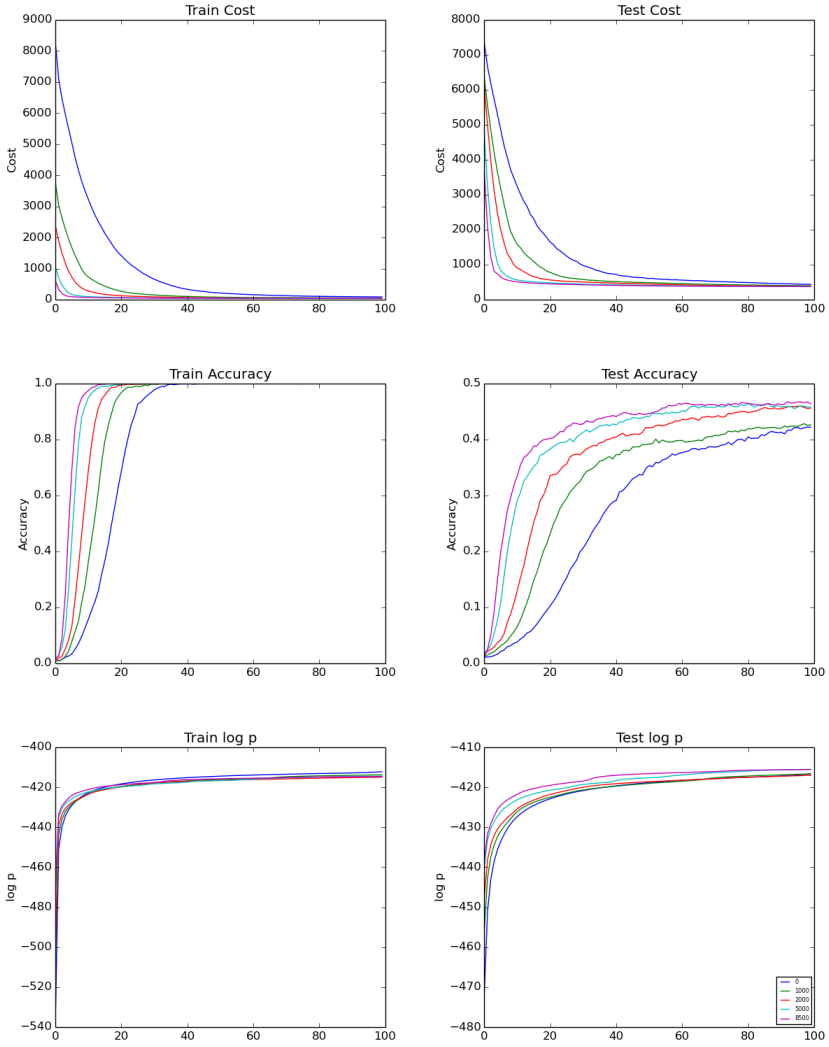


Figure 5.7: Training of the semi-supervised Fisher vector network on UCF-101, 64 GMM components, using 1000 labeled samples and changing the number of unlabeled data. Learning rate 0.001, 5 M-steps, $\lambda = 0.1$.

Chapter 6

Conclusions

Contents

In this thesis we have explored different deep learning architectures for the problem of action recognition in image sequences. Each of the three main chapters described a different approach for extracting representations of given videos; an unsupervised, a supervised and a semi-supervised approach have been proposed, all three focusing on incorporating the popular Fisher vector pipeline into the framework of deep neural networks.

We started by looking into using convolutional restricted Boltzmann machines as local feature extractors that can be trained in an unsupervised way in Chapter 3. Our main goal here was to see whether we could use the features that are learned straight from the data to replace the classic hand-crafted features usually used in the Fisher vector pipeline, and to see how their performance would compare when applied at the problem of action recognition. We have shown that even using features extracted from a shallow, single layer convolutional RBM network resulted in a higher classification performance than when handcrafted HOG features were used. However, we had trouble improving the performance by adding additional convolutional RBM layers. We argue that this is due to the complexity of choosing the right combination

of hyperparameters required for making the training work well.

We also modified the energy function of the convolutional RBM in a way that allowed us to use automatic differentiation tools when training the model, without having to do any additional normalizations of the learning updates. By implementing the convolutional RBM model using the Theano library, we managed to achieve a significant speedup when compared to the baseline Matlab implementation. Using our new implementation resulted in a 11x lower training time when the experiment was run on a CPU and a 76x lower training time when a GPU was used.

In order to come closer to the state of the art results in the problem of action recognition, deeper architectures need to be used than the one we explored in the first main chapter. For this reason we moved on to exploring other, larger deep convolutional neural networks for extracting local features from the input data. We replaced the single layer feature extraction part of our model with a larger network pretrained on the ImageNet image dataset which resulted in a large but expected boost of the classification accuracy.

We then defined the whole Fisher vector pipeline in terms of neural network layers, allowing us to view the whole pipeline as a single network that can be finetuned in an end to end fashion, leading to further significant performance improvements. We also showed that our network requires only a fraction of the number of trainable parameters needed when compared to the classical convolutional neural networks, while showing a comparable or better classification performance.

Finally, we have defined a hybrid network that combines unsupervised and supervised objective functions into a single joint objective function, forming a network that can leverage unlabeled data for the goal of increasing the model's classification performance. We performed a number of experiments that justified our reasoning and showed that increasing the amount of unlabeled data really resulted in improved per-

formance. This was shown on two different problems; an image classification problem on the CIFAR-10 dataset and on an action recognition problem on the UCF-101 dataset. These are promising results and further experiments should be performed to see how well the semi-supervised Fisher vector network would perform on the whole UCF-101 dataset when unlabeled data is added from a larger dataset, e.g. the YouTube-8M dataset [1].

6.1 Future work

Although recently the focus has moved away from unsupervised models such as the restricted Boltzmann machine towards deeper neural networks that are trained using supervision, we believe that it is worth further exploring unsupervised methods and architectures not only for the action recognition problem, but for other computer vision tasks. A sufficient enough motivation for this, as already mentioned in the introduction, should be the large and rapidly growing amount of image, video and audio data available online. Also, borrowing inspiration from biology, not everything we see around us comes with a label, yet we still manage to learn and understand our surroundings. The architecture we explored in Chapter 3 was too shallow to extract competitive features compared to those learned by the deeper network we later applied, so the first step would be to build deeper models that can be trained unsupervised.

The next step when talking about the convolutional Fisher vector network that we proposed in Chapter 4 and the action recognition problem would be to explore ways of adding motion information at the input. The straightforward way would be to train on pre-extracted optical flow, which is something we only briefly touched in our work. This could also be combined with a different network that computes the optical flow, as the network that we proposed is fairly flexible in terms of what can be used as input. Also worth exploring further is the idea of replacing fully connected layers with our proposed GMM and Fisher vector layers. Further experiments are required in order to

check whether this can always be done while still keeping a similar performance to the original network with fully connected layers. The application of our proposed network is not limited to the action recognition problem - it can be applied in a simple way to image classification problems or other tasks that deal with video data. We are already working on applying it to analyzing patient symptoms in video recordings.

The semi-supervised Fisher vector network that we proposed in Chapter 5 should also be researched further. One problem that we mentioned concerning the proposed network is the problem of learning trivial solutions when the GMM part is trained in an unsupervised way, only by maximizing the log-likelihood of the data. One way of overcoming this would be to combine it with an auto-encoder like objective, that is by also including a reconstruction term in the optimization. In terms of the application, the same conclusions apply as with the convolutional Fisher vector described in the preceding chapter. We note that we only used randomly sampled descriptors when performing the action recognition experiments, the architecture can easily be extended to support dense sampling, which should lead to better performance. The Gaussian mixture layer that we use in the architecture could also be included in different kinds of deep learning architectures and it might be useful in many different applications. One application where we intend to include the GMM layer is for human pose recognition.

Appendix A

Convolutional RBM Free energy and objective function updates

Contents

A.1 Deriving the Free energy term

To derive the Free energy of a Convolutional RBM, we have to marginalize over all of its hidden units. Starting from Equation 3.10, we write

$$
\begin{aligned}
p_\theta(\mathbf{v}) &= \frac{1}{Z}\sum_{\mathbf{h}} e^{-E_\theta(\mathbf{v},\mathbf{h})} = \\
&= \frac{1}{Z} e^{-\frac{1}{2D_v\sigma^2}\sum_{ch}\sum_{r,s}\left(v_{r,s,ch}-c_{ch}\right)^2} \cdot \sum_{\mathbf{h}} e^{\left(\sum_k \sum_{B_i,B_j} h_{i',j'} \sum_{i',j'\in B_{B_i,B_j}} h^k_{i',j'}\cdot I\left(h^k_{i',j'}\right)\right)^{\frac{1}{D_h}}} = \\
&= \frac{1}{Z} e^{\alpha} \cdot \left(\sum_{\mathbf{h}}\prod_k \prod_{B_i,B_j} e^{h_{i',j'}\sum_{\in B_{B_i,B_j}} h^k_{i',j'}\cdot I\left(h^k_{i',j'}\right)}\right)^{\frac{1}{D_h}} = \\
&= \frac{1}{Z} e^{\alpha} \cdot \prod_k \prod_{B_i,B_j}\left(\sum_{\mathbf{h}_{B_i,B_j}} e^{h_{i',j'}\sum_{\in B_{B_i,B_j}} h^k_{i',j'}\cdot I\left(h^k_{i',j'}\right)}\right)^{\frac{1}{D_h}} = \\
&= \frac{1}{Z} e^{\alpha} \cdot \prod_k \prod_{B_i,B_j}\left(1+\sum_{i',j'\in B_{B_i,B_j}} e^{I\left(h^k_{i',j'}\right)}\right)^{\frac{1}{D_h}} = \\
&= \frac{1}{Z} e^{\alpha} \cdot \prod_k \prod_{B_i,B_j} e^{\frac{1}{D_h}\log\left(1+\sum_{i',j'\in B_{B_i,B_j}} e^{I\left(h^k_{i',j'}\right)}\right)} = \\
&= \frac{1}{Z} e^{\alpha+\frac{1}{D_h}\sum_k\sum_{B_i,B_j}\log\left(1+\sum_{i',j'\in B_{B_i,B_j}} e^{I\left(h^k_{i',j'}\right)}\right)} = \\
&= \frac{1}{Z} e^{-F_\theta(\mathbf{v})}.
\end{aligned}
\tag{A.1}
$$

From the above we can see that the Convolutional RBM with Gaussian visible units has the Free energy defined as:

$$
\begin{aligned}
F_\theta(\mathbf{v}) = &\frac{1}{2D_v\sigma^2}\sum_{ch}\sum_{r,s}\left(v_{r,s,ch}-c_{ch}\right)^2 \\
&- \frac{1}{D_h}\sum_k\sum_{B_i,B_j}\log\left(1+\sum_{i',j'\in B_{B_i,B_j}} e^{I\left(h^k_{i',j'}\right)}\right),
\end{aligned}
\tag{A.2}
$$

using $I\left(h^k_{i,j}\right)$ as defined in Equation 3.4. If the visible units are binary, *i.e.* Bernoulli units, then we can remove the term summing over the squared visible units in Equation

A.2.

Numerical stability

Calculating the second term of the Free energy (Equation A.2) involves exponentiating the signal $I\left(h^k_{i',j'}\right)$, which can lead to numerical stability problems in the implementation. In order to bypass this problem and avoid getting infinities or *NaN*s we can modify our Free energy equation in the following way:

$$\begin{aligned} F_\theta(\mathbf{v}) &= \frac{1}{2D_v\sigma^2}\sum_{ch}\sum_{r,s}\left(v_{r,s,ch}-c_{ch}\right)^2 \\ &\quad -\frac{1}{D_h}\sum_k\sum_{B_i,B_j}\log\left(1+\sum_{i',j'\in B_{B_i,B_j}} e^{I\left(h^k_{i',j'}\right)}\right) = \\ &= \frac{1}{2D_v\sigma^2}\sum_{ch}\sum_{r,s}\left(v_{r,s,ch}-c_{ch}\right)^2 \\ &\quad -\frac{1}{D_h}\sum_k\sum_{B_i,B_j}\left(\beta_{B_i,B_j}+\log\left(e^{-\beta_{B_i,B_j}}+\sum_{i',j'\in B_{B_i,B_j}} e^{I\left(h^k_{i',j'}\right)-\beta_{B_i,B_j}}\right)\right), \end{aligned} \tag{A.3}$$

with β_{B_i,B_j} being the largest value of the signal in the block B_{B_i,B_j}:

$$\beta_{B_i,B_j} = \max_{i',j'\in B_{B_i,B_j}} I\left(h^k_{i',j'}\right). \tag{A.4}$$

Here we also used

$$\begin{aligned} \log\left(1+e^a+e^b+e^c\right) &= \log\left(e^0+e^a+e^b+e^c\right) = \\ &= \log\left(e^x\cdot\left(e^{0-x}+e^{a-x}+e^{b-x}+e^{c-x}\right)\right) = \\ &= \log\left(e^x\right)+\log\left(e^{-x}+e^{a-x}+e^{b-x}+e^{c-x}\right) = \\ &= x+\log\left(e^{-x}+e^{a-x}+e^{b-x}+e^{c-x}\right). \end{aligned} \tag{A.5}$$

A.2 Objective function updates

In this subsection we derive the gradients of the Free energy function needed to perform the training of the model. We start by taking the derivatives of $I\left(h^k_{i',j'}\right)$ with respect

to the filter weights:

$$\begin{aligned}\frac{\partial}{\partial w_{r,s}^k} I\left(h_{i',j'}^k\right) &= \frac{\partial}{\partial w_{r,s}^k}\left(\frac{1}{\sigma^2}\left(\tilde{\mathbf{W}}^k * \mathbf{v}\right)_{i',j'} + b_k\right) = \\ &= \frac{1}{\sigma^2} v_{i'+r,j'+s}\end{aligned} \tag{A.6}$$

The partial derivatives of the Free energy function $F_\theta(\mathbf{v})$ with respect to each weight $w_{r,s}^k$ of the filter k are calculated as:

$$\begin{aligned}\frac{\partial F_\theta(\mathbf{v})}{\partial w_{r,s}^k} &= \frac{\partial}{\partial w_{r,s}^k}\left(\frac{1}{2D_v\sigma^2}\sum_{ch}\sum_{r,s}\left(v_{r,s,ch}-c_{ch}\right)^2\right. \\ &\quad \left.-\frac{1}{D_h}\sum_{k'}\sum_{B_i,B_j}\log\left(1+\sum_{i',j'\in B_{B_i,B_j}} e^{I\left(h_{i',j'}^{k'}\right)}\right)\right) = \\ &= \frac{\partial}{\partial w_{r,s}^k}\left(-\frac{1}{D_h}\sum_{k'}\sum_{B_i,B_j}\log\left(1+\sum_{i',j'\in B_{B_i,B_j}} e^{I\left(h_{i',j'}^{k'}\right)}\right)\right) = \\ &= \frac{\partial}{\partial w_{r,s}^k}\left(-\frac{1}{D_h}\sum_{B_i,B_j}\log\left(1+\sum_{i',j'\in B_{B_i,B_j}} e^{I\left(h_{i',j'}^{k}\right)}\right)\right) = \\ &= -\frac{1}{D_h}\sum_{B_i,B_j}\left(\frac{1}{1+\sum_{i',j'\in B_{B_i,B_j}} e^{I\left(h_{i',j'}^{k}\right)}}\cdot\frac{\partial}{\partial w_{r,s}^k}\left(1+\sum_{i',j'\in B_{B_i,B_j}} e^{I\left(h_{i',j'}^{k}\right)}\right)\right) = \\ &= -\frac{1}{D_h}\sum_{B_i,B_j}\left(\frac{\sum_{i',j'\in B_{B_i,B_j}} e^{I\left(h_{i',j'}^{k}\right)}\cdot\frac{\partial}{\partial w_{r,s}^k} I\left(h_{i',j'}^{k}\right)}{1+\sum_{i',j'\in B_{B_i,B_j}} e^{I\left(h_{i',j'}^{k}\right)}}\right) = \\ &= -\frac{1}{D_h}\sum_{B_i,B_j}\left(\frac{\sum_{i',j'\in B_{B_i,B_j}} e^{I\left(h_{i',j'}^{k}\right)}\cdot\frac{1}{\sigma^2} v_{i'+r,j'+s}}{1+\sum_{i',j'\in B_{B_i,B_j}} e^{I\left(h_{i',j'}^{k}\right)}}\right) = \\ &= -\frac{1}{D_h\sigma^2}\sum_{B_i,B_j}\sum_{i',j'\in B_{B_i,B_j}} p\left(h_{i',j'}^k=1|\mathbf{v}\right) v_{i'+r,j'+s}\end{aligned} \tag{A.7}$$

Similarly for the hidden biases b_k we can write:

$$\frac{\partial F_\theta(\mathbf{v})}{\partial b^k} = -\frac{1}{D_h} \sum_{B_i, B_j} \sum_{i', j' \in B_{B_i, B_j}} p\left(h^k_{i', j'} = 1 | \mathbf{v}\right). \tag{A.8}$$

The partial derivatives with respect to the visible biases are calculated as:

$$\begin{aligned}
\frac{\partial F_\theta(\mathbf{v})}{\partial c_{ch}} &= \frac{\partial}{\partial c_{ch}} \left(\frac{1}{2 D_v \sigma^2} \sum_{ch} \sum_{r,s} \left(v_{r,s,ch} - c_{ch}\right)^2 \right) = \\
&= \frac{1}{2 D_v \sigma^2} \frac{\partial}{\partial c_{ch}} \left(\sum_{ch} \sum_{r,s} \left(v^2_{r,s,ch} - 2 v_{r,s,ch} c_{ch} + c^2_{ch}\right) \right) = \\
&= -\frac{1}{2 D_v \sigma^2} \sum_{r,s} 2 v_{r,s,ch} - 2 c_{ch} = \\
&= -\frac{1}{D_v \sigma^2} \sum_{r,s} v_{r,s,ch} - c_{ch}
\end{aligned} \tag{A.9}$$

After knowing all the derivatives derived above, every parameter of our model θ at time step $t+1$ is updated using the minibatch gradient descent with momentum update rule:

$$v^{t+1} = \gamma v^t + \mu \frac{1}{m} \sum_k^m \frac{\partial f^{reg}_\theta(\mathbf{v}_k)}{\partial \theta}$$

$$\theta^{t+1} = \theta^t - v^{t+1}, \tag{A.10}$$

where f^{reg}_θ is our regularized objective function introduced later in Equation A.14, γ is the momentum, μ is the learning rate and m is the size of the minibatch.

A.2.1 Defining the regularized cost function

As we defined earlier, the cost that we want to minimize is in fact the difference between two Free energies, the first one being the Free energy of the current training sample $\mathbf{v}$, and the second one the Free energy of the negative sample $\tilde{\mathbf{v}}$:

$$f_\theta = F_\theta(\mathbf{v}) - F_\theta(\tilde{\mathbf{v}}). \tag{A.11}$$

In order to keep the weights of the learnt filters small and to prevent overfitting, we regularize the cost function by adding a L2 norm regularization term for the weights

$$R_{L2} = ||\mathbf{W}||_2^2 = \left(\sqrt{\sum_{k,ch,i,j} w_{k,ch,i,j}^2} \right)^2 = \sum_{k,ch,i,j} w_{k,ch,i,j}^2. \tag{A.12}$$

Following the work of [57] we add a second regularization term which will force the activations of the hidden layers to be sparse. This is achieved by keeping the mean activation of the hidden layers close to a predefined target sparsity value p:

$$R_{sparsity} = \frac{1}{mK} \sum_{l=1}^{m} \left(p - \frac{1}{H_h H_w} \sum_{i,j} p\left(h_{i,j}^k | \mathbf{v}^{(l)} \right) \right)^2. \tag{A.13}$$

The regularized cost function can then be written as

$$f_\theta^{reg} = F_\theta(\mathbf{v}) - F_\theta(\tilde{\mathbf{v}}) + \lambda_{L2} R_{L2} + \lambda_{sparsity} R_{sparsity}. \tag{A.14}$$

Appendix **B**

Used datasets

Contents

B.1 UCF-101

The UCF-101 [93] dataset is one of the most challenging action recognition datasets today, consisting of 13320 realistic action videos collected from YouTube. It contains 101 different action categories that can be grouped into five different types: 1)Human-Object Interaction 2) Body-Motion Only 3) Human-Human Interaction 4) Playing Musical Instruments 5) Sports. The variety of the actions can be seen in Figure B.1. The videos of every action are divided into 25 groups containing $4-7$ videos, where every group shares common actors or background. All the videos have the same frame rate of 25 FPS and the resolution of 320×240 pixels. The mean duration of the videos is 7.21 seconds, with the shortest one being 1.06 seconds long, and the longest 71.04 seconds long. The total duration of the dataset summs up to 1600 minutes.

In order to standardize the experiment setup used when reporting results for the dataset, [93] provides three different predefined train/test splits that do not contain videos from the same groups in the training and testing sets. The final accuracy

that should be reported is the mean accuracy achieved for all three of the provided train/test splits.

Figure B.1: Example frames taken from videos from the UCF-101 dataset. The dataset contains videos from 101 different categories which can be grouped into five different types (Human-Object Interaction, Body-Motion Only, Human-Human Interaction, Playing Musical Instruments and Sports), here displayed with different border colours. Figure borrowed from [93].

B.1.1 State of the art results on UCF-101

B.2 CIFAR-10

The CIFAR-10 dataset [49] consists of 50 000 training and 10 000 testing RGB images of size 32×32 px, grouped into 10 different, mutually exclusive classes. The classes are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. Some examples of the images from the dataset are shown in Figure B.2.

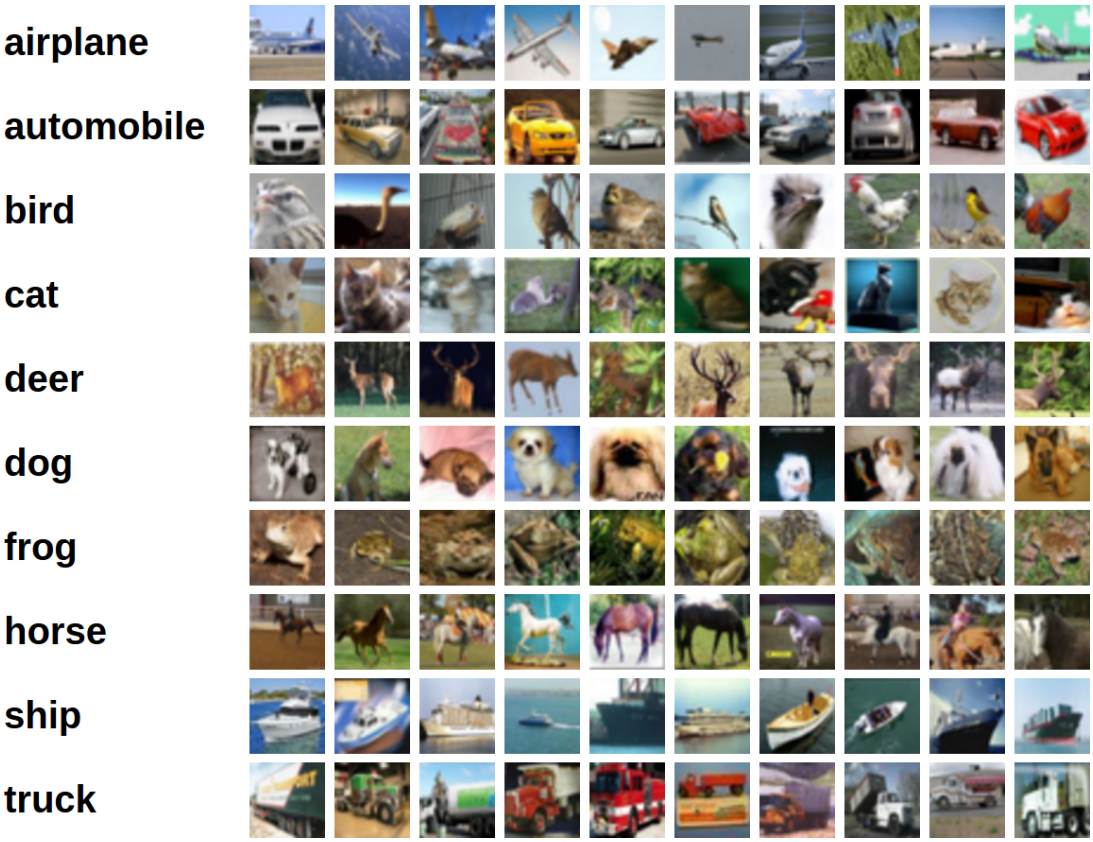


Figure B.2: Example images taken from the CIFAR-10 dataset. The dataset contains 60 000 RGB images of size 32×32 px from 10 different classes.

Table B.1: State of the art evaluated on UCF-101.

Method	**Split**	**Comment**	**Accuracy**
Slow fusion network [46]	all	trained on UCF101	41.3%
ResNet-18	split 1	reported in [70], trained on UCF101	47.3%
Sequential Verification [67]	split 1	reported in [70]	50.9%
VGAN [104]	split 1	reported in [70]	52.1%
Spatial stream [89]	split 1	trained on UCF101	52.3%
Slow fusion network [46]	all	pretrained on Sports1M	65.4%
What do objects tell [41]	all	objects	65.6%
ActionFlowNet [70]	split 1	trained on UCF101	70.0%
AlexNet on static RGB	all	from [13]	70.1%
Dynamic image network [13]	all	MDI end to end	70.6%
ActionFlowNet [70]	split 1	pretrained on FlyingChairs	71.0%
Trajectories [107]	all	HOG, from [108]	72.4%
Spatial stream [89]	split 1	pretrained on ImageNet	72.8%
Spatial stream [89]	all	pretrained on ImageNet	73.0%
VGG-M-2048 spatial [89]	split 1	reported in [27]	74.22%
Trajectories [107]	all	HOF, from [108]	76.0%
Dynamic image network [13]	all	MDI + RGB	76.9%
VGG-16	split 1	static RGB, from [110]	80.0%
ResNet-18	split 1	reported in [70], pretrained on ImageNet	80.7%
Trajectories [107]	all	MBH, from [108]	80.8%
Spatial net conv5 [111]	all		80.9%
Temporal stream [89]	split 1	10 frames optical flow	81.0%
Temporal net conv3 [111]	all		81.7%
Temporal net conv4 [111]	all		80.1%
Spatial net conv4 [111]	all		81.9%
Temporal conv3+conv4 [111]	all		82.2%
VGG-M-2048 temporal [89]	split 1	reported in [27], pretrained on ImageNet	82.34%
VGG-16 spatial	split 1	reported in [27]	82.61%
Spatial net conv4+conv5 [111]	all		82.8%
Temporal stream [89]	all		83.7%
What do objects tell [41]	all	motion	84.2%
Trajectories [107]	all	HOG+HOF+MBH, from [108]	84.8%
VGG-M-2048 late fusion [27]	split 1	pretrained on ImageNet	85.94%
VGG-16 temporal	split 1	reported in [27], pretrained on ImageNet	86.25%
Two stream network [89]	split 1		87.0%
Two stream network [89]	all		88.0%
What do objects tell [41]	all	objects + motion	88.1%
Dynamic image network [13]	all	MDI + RGB + trajectories	89.1%
TDD [111]	all	trajectory pooled deep-convolutional	90.3%
VGG-16 late fusion [27]	split 1	pretrained on ImageNet	90.62%
TDD + iDT [111]	all	trajectory pooled deep-convolutional	91.5%
Two stream [27]	all		92.5%
Two stream + iDT [27]	all		93.5%

Appendix C

Used architectures

Contents

C.1 CNN-M-2048

The structure of the CNN-M-2048 network introduced in [15] and used in [89] is shown in Table C.1.

C.1.1 Parameter count

The dimensionality of the last pooling layer in the CNN-M-2048 architecture (shown in Table C.1) for a single input is (512, 6, 6). The pooling layer is fully connected to 4096 units, followed by two more fully connected layers containing 2048 and 1000 units respectively. This corresponds to having 512 * 6 * 6 * 4096 + 4096 + 4096 * 2048 + 2048 + 2048 * 1000 + 1000 = 85 941 224 trainable parameters after the convolution and pooling layers. In case of the last fully connected layer having only 101 units, the parameter count is 512 * 6 * 6 * 4096 + 4096+ 4096 * 2048 + 2048 + 2048 * 101 + 101 = 84 099 173.

C.1.2 Spatial stream training

The training of the network as described in [89] uses stochastic gradient descent with momentum (set to 0.9) and minibatch size of 256. At each iteration, 256 frames are randomly sampled from the training videos which are rescaled so that the smaller side of the frames is 256 px. A 224×224 px sub-image is randomly cropped from each selected frame which then undergoes random horizontal flipping and RGB jittering.

C.1.3 Spatial stream testing

Given a video, a fixed number of frames is sampled with equal temporal spacing between them. From each frame 5 crops are extracted and also flipped, resulting in 10 crops that are fed into the network. The final class is picked based on the average score from the extracted crops.

Table C.1: The VGG-M-2048 architecture used in [89].

Layer	**Description**	**Shape**
input layer	-	(mb, 3, 224, 224)
conv1	96 filters 7×7px, stride 2px, pad 0px, LRN	(mb, 96, 109, 109)
pool1	2×2	(mb, 96, 54, 54)
conv2	256 filters 5×5px, stride 2px, pad 1px, LRN	(mb, 256, 26, 26)
pool2	2×2	(mb, 256, 13, 13)
conv3	512 filters 3×3px, stride 1px, pad 1px	(mb, 512, 13, 13)
conv4	512 filters 3×3px, stride 1px, pad 1px	(mb, 512, 13, 13)
conv5	512 filters 3×3px, stride 1px, pad 1px	(mb, 512, 13, 13)
pool3	2×2	(mb, 512, 6, 6)
fc6	fully connected	(mb, 4096)
fc7	fully connected	(mb, 2048)
fc8	fully connected	(mb, 1000)

C.2 VGG-16

The structure of the original VGG-16 [90] network is shown in Table C.2.

Table C.2: VGG-16 architecture.

Layer	Description	Shape	In our network
input layer	-	(mb, 3, 224, 224)	(mb, 3, 240, 320)
conv1_1 layer	64 filters 3×3px	(mb, 64, 224, 224)	(mb, 64, 240, 320)
conv1_2 layer	64 filters 3×3px	(mb, 64, 224, 224)	(mb, 64, 240, 320)
pool1	2×2	(mb, 64, 112, 112)	(mb, 64, 120, 160)
conv2_1 layer	128 filters 3×3px	(mb, 128, 112, 112)	(mb, 128, 120, 160)
conv2_2 layer	128 filters 3×3px	(mb, 128, 112, 112)	(mb, 128, 120, 160)
pool2	2×2	(mb, 128, 56, 56)	(mb, 128, 60, 80)
conv3_1 layer	256 filters 3×3px	(mb, 256, 56, 56)	(mb, 256, 60, 80)
conv3_2 layer	256 filters 3×3px	(mb, 256, 56, 56)	(mb, 256, 60, 80)
conv3_3 layer	256 filters 3×3px	(mb, 256, 56, 56)	(mb, 256, 60, 80)
pool3	2×2	(mb, 256, 28, 28)	(mb, 256, 30, 40)
conv4_1 layer	512 filters 3×3px	(mb, 512, 28, 28)	(mb, 512, 30, 40)
conv4_2 layer	512 filters 3×3px	(mb, 512, 28, 28)	(mb, 512, 30, 40)
conv4_3 layer	512 filters 3×3px	(mb, 512, 28, 28)	(mb, 512, 30, 40)
pool4	2×2	(mb, 512, 14, 14)	(mb, 512, 15, 20)
conv5_1 layer	512 filters 3×3px	(mb, 512, 14, 14)	(mb, 512, 15, 20)
conv5_2 layer	512 filters 3×3px	(mb, 512, 14, 14)	(mb, 512, 15, 20)
conv5_3 layer	512 filters 3×3px	(mb, 512, 14, 14)	(mb, 512, 15, 20)
pool5	2×2	(mb, 512, 7, 7)	-
fc6	fully connected	(mb, 4096)	-
fc7	fully connected	(mb, 4096)	-
fc8	fully connected	(mb, 1000)	-
softmax layer		(mb, 1000)	-

C.2.1 Parameter count

The dimensionality of the last pooling layer in the VGG-16 architecture (shown in Table C.2) for a single input is (512, 7, 7). The pooling layer is fully connected to 4096 units, followed by two more fully connected layers containing 4096 and 1000 units respectively. Including the biases, this corresponds to having 512 * 7 * 7 * 4096 + 4096 + 4096 * 4096 + 4096 + 4096 * 1000 + 1000 = 123 642 856 trainable parameters after the convolution and pooling layers. In case of the last fully connected layer having only 101 units, the parameter count is 512 * 7 * 7 * 4096 + 4096 + 4096 * 4096 + 4096 + 4096 * 101 + 101 = 119 959 653.

C.2.2 Spatial stream training

As reported in [27], the same procedure for training the spatial stream is used as the one described in [89], with small differences; no RGB jittering is used and the learning rate is not decreased based on a fixed schedule, but it is lowered when the validation error saturates.

Bibliography

[1] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan. Youtube-8m: A large-scale video classification benchmark. *arXiv preprint arXiv:1609.08675*, 2016. 83, 88

[2] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for Boltzmann Machines*. *Cognitive science*, 9(1):147–169, 1985. 16

[3] R. Arandjelovic and A. Zisserman. All about VLAD. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1578–1585, 2013. 15

[4] I. Arel, D. C. Rose, and T. P. Karnowski. Deep machine learning-a new frontier in artificial intelligence research [research frontier]. *Computational Intelligence Magazine, IEEE*, 5(4):13–18, 2010. 3

[5] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007. 68, 76

[6] I. Atmosukarto, B. Ghanem, and N. Ahuja. Trajectory-based Fisher kernel representation for action recognition in videos. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 3333–3336. IEEE, 2012. 12

[7] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt. Sequential deep learning for human action recognition. In *Human Behavior Understanding*, pages 29–39. Springer, 2011. 19

[8] M. A. Bagheri, Q. Gao, S. Escalera, A. Clapes, K. Nasrollahi, M. B. Holte, and T. B. Moeslund. Keep it accurate and diverse: Enhancing action recognition

performance by ensemble learning. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2015 IEEE Conference on*, pages 22 – 29, 2015. 38

[9] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. *Computer vision–ECCV 2006*, pages 404–417, 2006. 12

[10] Y. Bengio. Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009. 3, 28, 30

[11] Y. Bengio. Deep learning of representations: Looking forward. *arXiv preprint arXiv:1305.0445*, 2013. 3

[12] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007. 3, 16

[13] H. Bilen, B. Fernando, E. Gavves, A. Vedaldi, and S. Gould. Dynamic image networks for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3034–3042, 2016. 99

[14] K. Chatfield, V. S. Lempitsky, A. Vedaldi, and A. Zisserman. The devil is in the details: an evaluation of recent feature encoding methods. In *BMVC*, volume 2, page 8, 2011. 15

[15] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014. 4, 19, 54, 100

[16] B. Chen, J.-A. Ting, B. Marlin, and N. de Freitas. Deep learning of invariant spatio-temporal features from video. *NIPS 2010 Deep Learning and Unsupervised Feature Learning Workshop*, 2010. 18

[17] G. Cheng, Y. Wan, A. N. Saudagar, K. Namuduri, and B. P. Buckles. Advances in human action recognition: A survey. *arXiv preprint arXiv:1501.05964*, 2015. 10

[18] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012. 4

[19] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *Workshop on statistical learning in computer vision, ECCV*, volume 1, pages 1–2. Prague, 2004. 13

[20] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005. 12

[21] N. Dalal, B. Triggs, and C. Schmid. Human detection using oriented histograms of flow and appearance. In *European conference on computer vision*, pages 428–441. Springer, 2006. 13

[22] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977. 64, 65, 66, 67

[23] S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. K. Sønderby, D. Nouri, D. Maturana, M. Thoma, E. Battenberg, J. Kelly, J. D. Fauw, M. Heilman, D. M. de Almeida, B. McFee, H. Weideman, G. Takács, P. de Rivaz, J. Crall, G. Sanders, K. Rasul, C. Liu, G. French, and J. Degrave. Lasagne: First release., Aug. 2015. 55

[24] P. Dollár, V. Rabaud, G. Cottrell, and S. Belongie. Behavior recognition via sparse spatio-temporal features. In *Visual Surveillance and Performance Evalu-*

ation of Tracking and Surveillance, 2005. 2nd Joint IEEE International Workshop on, pages 65–72. IEEE, 2005. 11, 12, 14

[25] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011. 55, 67

[26] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, 11:625–660, 2010. 3

[27] C. Feichtenhofer, A. Pinz, and A. Zisserman. Convolutional two-stream network fusion for video action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016*, pages 1933–1941, 2016. 4, 19, 56, 59, 60, 99, 103

[28] A. Fischer and C. Igel. An introduction to restricted Boltzmann machines. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pages 14–36. Springer, 2012. 3, 16

[29] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 12

[30] Y. Freund and D. Haussler. *Unsupervised learning of distributions of binary vectors using two layer networks*. Computer Research Laboratory [University of California, Santa Cruz], 1994. 16

[31] T. Guha and R. K. Ward. Learning sparse representations for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(8):1576–1588, 2012. 14

[32] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Manchester, UK, 1988. 11

[33] S. Herath, M. Harandi, and F. Porikli. Going deeper into action recognition: A survey. *Image and Vision Computing*, 60:4–21, 2017. 2, 10

[34] G. Hinton. Where do features come from? *Cognitive science*, 38(6):1078–1101, 2014. 3

[35] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002. 3, 16, 29, 30

[36] G. E. Hinton. A practical guide to training restricted Boltzmann machines. *Momentum*, 9(1), 2010. 3, 15, 16, 25, 28

[37] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006. 3, 16, 29

[38] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. 18

[39] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106, 1962. 3

[40] T. Jaakkola, D. Haussler, et al. Exploiting generative models in discriminative classifiers. *Advances in neural information processing systems*, pages 487–493, 1999. 14

[41] M. Jain, J. C. van Gemert, and C. G. Snoek. What do 15,000 object categories tell us about classifying and localizing actions? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 46–55, 2015. 59, 99

[42] P. Janssen, S. Kalkan, M. Lappe, A. Leonardis, J. Piater, L. Wiskott, et al. Deep hierarchies in the primate visual cortex: What can we learn for computer vision? *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2012. 3

[43] I. Jargalsaikhan, S. Little, C. Direkoglu, and N. E. O'Connor. Action recognition based on sparse motion trajectories. *Image Processing (ICIP), 2013 20th IEEE International Conference on*, 2013. 12

[44] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3304–3311. IEEE, 2010. 15

[45] S. Ji, W. Xu, M. Yang, and K. Yu. 3D convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 221–231, 2013. 4, 19

[46] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1725–1732. IEEE, 2014. 4, 19, 38, 39, 58, 59, 99

[47] A. Klaser, M. Marszałek, and C. Schmid. A spatio-temporal descriptor based on 3D-gradients. In *BMVC 2008-19th British Machine Vision Conference*, pages 275–1. British Machine Vision Association, 2008. 12

[48] J. Krapac, J. Verbeek, and F. Jurie. Modeling spatial layout with Fisher vectors for image categorization. In *2011 International Conference on Computer Vision*, pages 1487–1494. IEEE, 2011. 48

[49] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009. 63, 75, 98

[50] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1106–1114, 2012. 4, 18

[51] I. Laptev. On space-time interest points. *International journal of computer vision*, 64(2-3):107–123, 2005. 11, 12

[52] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld. Learning realistic human actions from movies. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. 12, 14

[53] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, volume 2, pages 2169–2178. IEEE, 2006. 50

[54] Q. V. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. S. Corrado, J. Dean, and A. Y. Ng. Building high-level features using large scale unsupervised learning. *arXiv preprint arXiv:1112.6209*, 2011. 4

[55] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. 3

[56] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 17, 18

[57] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM, 2009. 3, 17, 24, 25, 26, 32, 33, 45, 55, 95

[58] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Unsupervised learning of hierarchical representations with convolutional deep belief networks. *Communications of the ACM*, 54(10):95–103, 2011. 7, 17, 23, 24, 25, 26, 31

[59] H. Lee, P. Pham, Y. Largman, and A. Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in neural information processing systems*, pages 1096–1104, 2009. 25

[60] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004. 11, 12

[61] B. D. Lucas, T. Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981. 11

[62] B. M. Marlin, K. Swersky, B. Chen, and N. D. Freitas. Inductive principles for restricted Boltzmann machine learning. In *International Conference on Artificial Intelligence and Statistics*, pages 509–516, 2010. 29

[63] M. Marszałek, I. Laptev, and C. Schmid. Actions in context. In *IEEE Conference on Computer Vision & Pattern Recognition*, 2009. 11

[64] P. Matikainen, M. Hebert, and R. Sukthankar. Trajectons: Action recognition through the motion analysis of tracked features. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 514–521. IEEE, 2009. 11

[65] R. Memisevic and G. E. Hinton. Unsupervised learning of image transformations. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007. 17

[66] R. Messing, C. Pal, and H. Kautz. Activity recognition using the velocity histories of tracked keypoints. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 104–111. IEEE, 2009. 11

[67] I. Misra, C. L. Zitnick, and M. Hebert. Shuffle and learn: unsupervised learning using temporal order verification. In *European Conference on Computer Vision*, pages 527–544. Springer, 2016. 99

[68] R. M. Neal and G. E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998. 65

[69] F. Negin and F. Bremond. Human action recognition in videos: A survey. Technical report, INRIA Technical Report, Sophia Antipolis, France, 2016. 10, 11, 12

[70] J. Y.-H. Ng, J. Choi, J. Neumann, and L. S. Davis. Actionflownet: Learning motion representation for action recognition. *arXiv preprint arXiv:1612.03052*, 2016. 99

[71] J. Y.-H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. *arXiv preprint arXiv:1503.08909*, 2015. 54

[72] J. Ngiam, Z. Chen, P. W. Koh, and A. Y. Ng. Learning deep energy models. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1105–1112, 2011. 16

[73] J. C. Niebles, H. Wang, and L. Fei-Fei. Unsupervised learning of human action categories using spatial-temporal words. *International journal of computer vision*, 79(3):299–318, 2008. 14

[74] B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision research*, 37(23):3311–3325, 1997. 14

[75] D. Oneata, J. Verbeek, and C. Schmid. Action and event recognition with Fisher vectors on a compact feature set. In *Proceedings of the IEEE international conference on computer vision*, pages 1817–1824, 2013. 15

[76] P. Palasek and I. Patras. Action recognition using convolutional restricted Boltzmann machines. In *Proceedings of the 1st International Workshop on Multimedia Analysis and Retrieval for Multimodal Interaction*, MARMI '16, pages 3–8, New York, NY, USA, 2016. ACM. 20, 45, 55, 57, 58

[77] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 67

[78] X. Peng, C. Zou, Y. Qiao, and Q. Peng. Action recognition with stacked Fisher vectors. In *European Conference on Computer Vision*, pages 581–595. Springer, 2014. 20

[79] F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007. 14

[80] F. Perronnin and D. Larlus. Fisher vectors meet neural networks: A hybrid classification architecture. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3743–3752, 2015. 20

[81] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid. Epicflow: Edge-preserving interpolation of correspondences for optical flow. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1164–1172, 2015. 57

[82] M. D. Rodriguez, J. Ahmed, and M. Shah. Action mach a spatio-temporal maximum average correlation height filter for action recognition. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. 11

[83] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 18

[84] G. Salton. Automatic information organization and retrieval. 1968. 13

[85] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek. Image classification with the Fisher vector: Theory and practice. *International journal of computer vision*, 105(3):222–245, 2013. 15, 36, 39, 47, 49

[86] C. Schuldt, I. Laptev, and B. Caputo. Recognizing human actions: a local SVM approach. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 3, pages 32–36. IEEE, 2004. 11

[87] P. Scovanner, S. Ali, and M. Shah. A 3-dimensional SIFT descriptor and its application to action recognition. In *Proceedings of the 15th ACM international conference on Multimedia*, pages 357–360. ACM, 2007. 12

[88] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep Fisher networks for large-scale image classification. In *Advances in neural information processing systems*, pages 163–171, 2013. 20

[89] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014. 4, 19, 38, 39, 54, 55, 58, 59, 60, 99, 100, 101, 103

[90] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015. 4, 19, 46, 53, 56, 59, 75, 101

[91] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. 1986. 16

[92] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart, J. L. McClelland, et al., editors, *Parallel Distributed Processing: Volume 1: Foundations*, pages 194–281. MIT Press, Cambridge, 1987. 3

[93] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012. 7, 23, 33, 55, 96, 97

[94] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. 57

[95] J. Sun, X. Wu, S. Yan, L.-F. Cheong, T.-S. Chua, and J. Li. Hierarchical spatio-temporal context modeling for action recognition. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2004–2011. IEEE, 2009. 11

[96] I. Sutskever and G. E. Hinton. Learning multilevel distributed representations for high-dimensional sequences. In *International Conference on Artificial Intelligence and Statistics*, pages 548–555, 2007. 17

[97] V. Sydorov, M. Sakurada, and C. H. Lampert. Deep Fisher kernels-end to end learning of the Fisher kernel GMM parameters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1402–1409, 2014. 20

[98] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 19

[99] G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler. Convolutional learning of spatio-temporal features. In *Computer Vision–ECCV 2010*, pages 140–153. Springer, 2010. 18

[100] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. 7, 23, 32, 55

[101] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2), 2012. 67

[102] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3D convolutional networks. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 4489–4497. IEEE, 2015. 4, 19

[103] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 9999:3371–3408, 2010. 3

[104] C. Vondrick, H. Pirsiavash, and A. Torralba. Generating videos with scene dynamics. In *Advances In Neural Information Processing Systems*, pages 613–621, 2016. 99

[105] H. Wang, A. Klaser, C. Schmid, and C.-L. Liu. Action recognition by dense trajectories. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3169–3176. IEEE, 2011. 12, 13, 20, 35, 36, 38, 39, 46

[106] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu. Dense trajectories and motion boundary descriptors for action recognition. *International Journal of Computer Vision*, pages 1–20, 2013. 12, 13, 55

[107] H. Wang and C. Schmid. Action recognition with improved trajectories. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 3551–3558. IEEE, 2013. 12, 20, 37, 38, 39, 99

[108] H. Wang and C. Schmid. LEAR-INRIA submission for the THUMOS workshop. In *ICCV Workshop on Action Recognition with a Large Number of Classes*, 2013. 38, 99

[109] H. Wang, M. M. Ullah, A. Klaser, I. Laptev, and C. Schmid. Evaluation of local spatio-temporal features for action recognition. In *BMVC 2009-British Machine Vision Conference*, pages 124–1. BMVA Press, 2009. 11, 12, 20

[110] J. Wang, A. Cherian, and F. Porikli. Ordered pooling of optical flow sequences for action recognition. *arXiv preprint arXiv:1701.03246*, 2017. 99

[111] L. Wang, Y. Qiao, and X. Tang. Action recognition with trajectory-pooled deep-convolutional descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4305–4314, 2015. 38, 39, 54, 99

[112] X. Wang, L. Wang, and Y. Qiao. A comparative study of encoding, pooling and normalization methods for action recognition. In *Asian Conference on Computer Vision*, pages 572–585. Springer, 2012. 15

[113] M. Welling, M. Rosen-Zvi, and G. E. Hinton. Exponential family harmoniums with an application to information retrieval. In *Advances in neural information processing systems*, pages 1481–1488, 2004. 16

[114] G. Willems, T. Tuytelaars, and L. Van Gool. An efficient dense and scale-invariant spatio-temporal interest point detector. *Computer Vision–ECCV 2008*, pages 650–663, 2008. 11

[115] Y. Yi and Y. Lin. Human action recognition with salient trajectories. *Signal Processing*, 2013. 12

[116] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4694–4702, 2015. 19

[117] S. Zhao, Y. Liu, Y. Han, and R. Hong. Pooling the convolutional layers in deep convnets for action recognition. *arXiv preprint arXiv:1511.02126*, 2015. 20, 38, 39, 54

[118] F. Zhu, L. Shao, J. Xie, and Y. Fang. From handcrafted to learned representations for human action recognition: A survey. *Image and Vision Computing*, 55:42–52, 2016. 10, 12

[119] Y. Zhu, X. Zhao, Y. Fu, and Y. Liu. Sparse coding on local spatial-temporal volumes for human action recognition. In *Asian Conference on Computer Vision*, pages 660–671. Springer, 2010. 14